CIRCUIT COMPLEXITY OF CONSTRAINT SATISFACTION PROBLEMS WITH FEW SUBPOWERS

by

Amir El-Aooiti

Bachelor of Science, Ryerson University, 2015

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the Degree of

Master of Science

in the program of

Applied Mathematics

Toronto, Ontario, Canada, 2017

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

CIRCUIT COMPLEXITY OF CONSTRAINT SATISFACTION PROBLEMS WITH FEW SUBPOWERS

Amir El-Aooiti
Master of Science, 2017
Applied Mathematics
Ryerson University

Although Constraint Satisfaction Problems (CSPs) are generally known to be **NP**-complete, placing restrictions on the constraint template can yield tractable subclasses. By studying the operations in the polymorphism of the constraint language, we can construct algorithms which solve our CSP in polynomial time. Previous results for CSPs with Mal'tsev [**7**] and generalized majority-minority operations [**10**] were improved to include CSPs with k-edge operations [**15**]. We present an alternative method to solve k-edge CSPs by utilizing Boolean trees placing the problem in the class $\mathbf{NC}^2$. We do this by arranging the logical formulas describing the CSP into a Boolean tree where each leaf represents a constraint in the CSP. We take the conjunction of the constraint formulas yielding partial solutions at every step until we are left with a solution set at the root of the tree which satisfies all of the constraints.

ACKNOWLEDGMENTS

بسم الله الرحمٰن الرحيم

To begin, I would like to thank Allah for giving me the opportunity to succeed. It is only by his will that I am able to accomplish my goals now and for the years to come.

No one has stood by me like my family. They have provided me with nothing short of total love and support during my life, and I have no doubt they will continue to support me long after. For that, they have my lifelong gratitude.

I am deeply appreciative of the faculty and staff who were part of this program. I would like to thank Dr. Bonato, Dr. Escobar, Dr. Ferrando, Dr. Ilie, Dr. Prałat, and Dr. Rohlf for the hours they spent lecturing; Dr Georgiou for conducting the seminars; and Luisa Chan, Steve Kanellis, Teresa Lee, and Kathy Peter, for their administrative and technical work.

I would like to acknowledge and thank the members of the thesis defense committee Dr. Bonato, Dr. Delić, Dr. Georgiou, and Dr. Pascal.

Most of all, I want to express my deep appreciation for my supervisor Dr. Dejan Delić for giving me the opportunity to write a fulfilling thesis and for guiding me the entire way.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION TO COMPLEXITY THEORY

An algorithm is generally defined as a series of steps taken to solve or verify a solution to a problem. Algorithms which solve mathematical problems require both space and time to produce and verify solutions and can be classified based on their efficiency. Complexity theory is the field of mathematics which classifies problems based on their characteristics and the requirements of an algorithm to produce or verify given solutions; problems are called “hard” if they cannot be solved “efficiently”. In this chapter, we will define and study the complexity classes **P**, **NP**, **NC**, **L**, and **NL**. Problems in these classes (among others) can be solved using what is called a *Turing machine*. Originally a real computer invented by British mathematician Alan Turing in 1936, a Turing machine is a mathematical model used to simulate the process an algorithm takes to solve or verify a solution to a problem. A Turing machine traverses a one-dimensional strip with symbols containing the input and manipulates the symbols in accordance with the algorithm being implemented. A modern computer today simulates the logical steps taken by a Turing machine, albeit much faster.

1.1 Polynomial Time Classes

In this section, we will study the polynomial time classes **P** and **NP**. These classes classify problems based on the amount of time, not space, taken by a Turing machine to solve or verify a solution to a problem.

Definition 1.1. Given an input string of size n, problems in **P** can be solved by a Turing machine in $\mathcal{O}(n^p)$-many steps for some positive integer p.

Problems in **P** are said to be solvable in polynomial time and are called *tractable*. Consider the following example of a tractable problem:

Example 1.1. The product of two $n \times n$ matrices can be computed in polynomially many steps in n. Suppose we are taking the product of $n \times n$ matrices A and B with entries $a_{i,j}$ and $b_{i,j}$ respectively, and let $AB = C$. The formula corresponding to each entry in C,

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j},$$

can be computed in n steps and the matrix C itself will have n^2 entries. The products themselves can be computed in polynomial time given that the matrix entries are not large. Hence, it takes a total of $\mathcal{O}(n^3)$-many steps to compute all of the entries in C using this formula.

Example 1.2. Other examples of problems in **P** include:

(1) basic arithmetic functions,

(2) Euclid's algorithm for finding the greatest common divisor,

(3) (s,t)-connectivity problem for graphs, and

(4) linear programming.

Next, we will consider the class **NP**, a class whose problems are said to be "harder" than those in **P**.

Definition 1.2. Given an input string of size n, nondeterministic solutions to problems in **NP** can be verified (but not necessarily solved) by a nondeterministic Turing machine in $\mathcal{O}(n^p)$-many steps for some positive integer p.

Problems in **NP** are said to be verifiable in polynomial time. If a problem in **NP** cannot be solved in polynomial time by any known algorithm, it is called *intractable.* Consider the following example of an intractable problem:

Example 1.3. Consider a graph $G = (V, E)$ for which we want to verify the existence of a Hamiltonian path. If we nondeterministically choose a permutation of vertices $v_1, v_2, ..., v_n \in V(G)$, then we can verify that it forms a Hamiltonian path by checking that $v_i v_{i+1}$ is an edge in $E(G)$ for all $1 \leq i \leq n-1$. Since there are $n-1$ consecutive vertex pairs in our permutation, it takes a total of $n-1 = \mathcal{O}(n)$ steps to verify that our permutation is a Hamiltonian path in G.

Example 1.4. Other examples of problems in **NP** include:

(1) finding the smallest dominating set on a graph,

(2) Boolean satisfiability,

(3) graph isomorphism, and

(4) all problems in **P**.

It is known that $\mathbf{P} \subseteq \mathbf{NP}$ because any problem which can be solved in polynomial time can also be verified in polynomial time. Note that **P** is not a proper subset of **NP** because it remains an open problem as to whether $\mathbf{P} = \mathbf{NP}$; it is conjectured and generally accepted that in fact $\mathbf{P} \neq \mathbf{NP}$. With this assumption, we can introduce two more complexity classes.

1.2 Completeness

Reductions are a central tool in complexity theory allowing us to further classify problems into subsets of **P** and **NP**. Given a Turing machine, a single valid input for which the Turing machine returns an affirmative result is called a *string*. A language L is the set of all strings accepted by a specific Turing machine. Referring to Example 1.3, suppose we are given a Turing machine which verifies that a graph has a Hamiltonian path. An acceptable input for this Turing machine is a graph G with a Hamiltonian path. The language accepted by this Turing machine is written as the language

$$\textsc{Hampath} = \{G : G \text{ has a Hamiltonian path}\}.$$

Definition 1.3. Given two languages L_1 and L_2, a *reduction* from L_1 to L_2 is a function $f : L_1 \longrightarrow L_2$ such that $x \in L_1$ if and only if $f(x) \in L_2$. Furthermore, if f itself can be computed in polynomially many steps in the size of the input string from L_1, then f is referred to as a polynomial-time reduction. If there exists a polynomial-time reduction from language L_1 to L_2, we write $L_1 \leq_P L_2$.

Reductions are useful when it comes to classifying problems into complexity classes. We can show that a problem belongs to a complexity class by reducing another problem from the same complexity class to it using a function with the appropriate complexity. With reductions, we can define two more subsets contained in **P** and **NP**.

Definition 1.4. A problem in **P** is called **P**-*complete* if all other problems in **P** can be reduced to it using a reduction from the class **NC** (see next section). Analogously, a problem in **NP** is called **NP**-*complete* if all other problems in **NP** can be reduced to it using a reduction from the class **P**.

Example 1.5. The Boolean satisfiability problem is the problem of deciding whether there exists an assignment of literals which satisfies a Boolean formula. A Boolean formula is said to be written in n-disjunctive normal form (n-DNF) if it is written as a disjunction of conjunctions which each contain n literals. The problem of finding an assignment of literals for a formula in 3-DNF is written as the language

$$\text{3-S}\textsc{at} = \{\Phi : \Phi \text{ is a satisfiable Boolean formula in 3-DNF}\}.$$

An *independent set* in a graph G is a set of vertices $S \subseteq V(G)$ for which no two vertices in S are adjacent. For some positive integer n, the problem of finding an independent set of size n in a graph is written as the language,

$$\textsc{Ind} = \{G : G \text{ has an independent set of size } n\}.$$

An instance of 3-Sat can be reduced to an instance of Ind in polynomial time. First we convert the Boolean formula Φ to conjunctive normal form with n clauses in order to create a new graph G. For each clause in Φ, we create a triangle graph where each vertex represents a literal for a total of $3n$ vertices. Then, we place edges between the triangles between vertices whose literals are negations of each other. This reduction can be done in polynomial time. It can be shown that finding an assignment of literals satisfying Φ is equivalent to finding the largest independent set on G by assigning "TRUE" to each vertex in the independent set. Hence, 3-Sat $\leq_P$ Ind.

Both the set of **P**-complete and **NP**-complete problems are contained in **P** and **NP** respectively. If we assume that $\mathbf{P} \neq \mathbf{NP}$, then it follows that $\mathbf{P} \cap (\mathbf{NP}\text{-complete}) = \emptyset$. As a result, **NP**-complete problems are harder than those in **NP** and the analogous statement is true regarding **P**-complete problems and problems in **P**.

1.3 Circuit Complexity Classes

The study of the circuit complexity of problems provides us with a new complexity class called **NC** (Nick's Class) named after computer scientist Dr. Nicholas Pippenger. While problems in **P** and **NP** are classified by the efficiency in which they are solved by a Turing machine, problems in NC are classified by the efficiency in which they are solved by a *Boolean circuit*; a wholly different model. A Boolean circuit (or Boolean tree) is a directed tree with a maximum vertex degree of 3. Vertices of degree 3 are labeled with either the binary $\wedge$ ("and") or $\vee$ ("or") operations, vertices of degree 2 are labeled with the unary $\neg$ ("not") operation, vertices of degree 1 are labeled as input gates, and a single vertex of degree 1 is labeled as an output gate. The *size* of a Boolean circuit is the number of input gates it contains and the *depth* of a Boolean circuit is the maximal length of a path from some input gate to the output gate. The circuit complexity of a problem depends not on the number of steps taken, but on the number of input gates in the Boolean circuit and its depth.

Definition 1.5. Given an input string of size n, problems in $\mathbf{NC}^i$ can be solved using a Boolean circuit which contains $\mathcal{O}(n^p)$-many input gates and has a depth of $\mathcal{O}(\log^i n)$ for some positive integers i and p. We define the class **NC** as

$$\mathbf{NC} = \bigcup_{i=1}^{\infty} \mathbf{NC}^i.$$

Example 1.6. A single entry of the product C of two $n \times n$ Boolean matrices A and B can be computed in $\mathbf{NC}^1$. The formula corresponding to each entry in C is

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

for binary entries $a_{i,j}$ and $b_{i,j}$ in A and B respectively. We can compute the conjunction $a_{i,k} \wedge b_{k,j}$ in depth 1 for each $k \in [n]$ and we can compute the disjunction of the n conjunction terms in depth $\mathcal{O}(\log n)$. In total, the formula for each entry can be calculated in depth $\mathcal{O}(\log n+1) = \mathcal{O}(\log n)$. Furthermore, since the computation of each entry involves taking a row from A and a column from B, the number of inputs in our Boolean circuit is equal to $2n = \mathcal{O}(n)$, a polynomial.

It is known that Boolean circuit algorithms can be simulated by a Turing machine, but the converse of that statement is still an open problem; hence, we have $\mathbf{NC} \subseteq \mathbf{P}$. It is conjectured and generally accepted that $\mathbf{NC} \neq \mathbf{P}$. It follows that $\mathbf{NC} \cap$ ($\mathbf{P}$-complete) $= \emptyset$. By the definition of $\mathbf{NC}$, we have $\mathbf{NC}^1 \subseteq \mathbf{NC}^2 \subseteq ... \subseteq \mathbf{NC}^i \subseteq \mathbf{NC}$ for any positive integer i. Although it is an open problem as to whether $\mathbf{NC}^i \neq \mathbf{NC}^{i+1}$ for all i, it is conjectured that this is the case.

1.4 Logspace Classes

Finally, we will study, but not delve into, the logspace classes **L** and **NL**. Logspace classes classify problems based on the space, not time, used by a Turing machine to solve or verify a solution to a problem. **L** and **NL** have an important fit into our complexity hierarchy discussed so far.

Definition 1.6. Given an input string of size n, problems in **L** can be solved by a Turing machine in $\mathcal{O}(\log^p n)$ amount of rewritable memory space and nondeterministic solutions to problems in **NL** can be verified by a nondeterministic Turing machine in the same space for some positive integer p.

It is known that $\mathbf{L} \subseteq \mathbf{NL}$ and it is conjectured, and generally accepted, that $\mathbf{L} \neq \mathbf{NL}$. Having defined these classes, we are ready to illustrate a hierarchy of space and time complexity. For any positive integer i, we can order our classes as follows:

$$\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}^2 \subseteq \mathbf{NC}^3 \subseteq ... \subseteq \mathbf{NC}^i \subseteq \mathbf{NC} \subseteq \mathbf{P} \subseteq \mathbf{NP}.$$

This ordering can also be seen as an order of hardness with the class $\mathbf{NC}^1$ being the easiest and $\mathbf{NP}$ being the hardest of the classes mentioned thus far. Many more classes have also been studied and can be added to our ordering on both ends and in between classes, but they go beyond the scope of this paper.

Chapter 2

CONSTRAINT SATISFACTION PROBLEMS

A constraint satisfaction problem (CSP) is a mathematical model used to express combinatorial problems for which a set of variables must be made to satisfy a set of constraints by assigning to them elements from a given domain. Many problems can be expressed in the language of CSPs and, thus, can all be solved using algorithms designed to solve CSPs. For the purposes of this paper, we will assume that all CSPs contain a finite domain and finitely many variables (unless mentioned otherwise) so as to produce finitely sized solutions. An algorithm which solves a CSP will either return a set of possible solutions in the form of functions, or return an empty set indicating that no assignment of variables satisfies the given constraints. While algorithms can be designed to output an optimal solution to a CSP, for our purposes, we will focus on finding all solutions satisfying the constraints including those which are suboptimal.

2.1 A Proper Definition of a CSP

A CSP can be interpreted as a mapping from a domain to a set of variables which satisfies the constraints imposed on each variable. Constraints may either be imposed artificially in order to find desirable or optimal solutions, or they may be naturally assumed so as to conform to physical boundaries and avoid logical fallacies. A CSP can be expressed as a mathematical object.

Definition 2.1. An instance of a CSP is a triple $(V, A, \mathcal{C})$ such that:

- V is a finite set of variables
- A is a finite set known as the domain
- $\mathcal{C}$ is a set of constraints $\{C_1, C_2, ..., C_q\}$, where each constraint is a pair $(\mathbf{s}_i, R_i)$ such that for all $i \in [q]$:
  - $\mathbf{s}_i \in V^{m_i}$ is a tuple of size m_i (the *scope* of C_i).
  - $R_i \subseteq A^{m_i}$ is a m_i-ary relation (the *template* of C_i).

The solution set of the CSP is the set of functions $f : V \longrightarrow A$ such that for all $i \in [q]$, we have $f(\mathbf{s}_i) \in R_i$.

Example 2.1. The k-Colourability problem for a graph G can be interpreted as the triple $(V, D, \{C_1, C_2, ..., C_{|E(G)|}\})$, where:

- $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices,
- $D = \{1, 2, ..., k\}$ the domain of colours, and
- $C_i = (\mathbf{e}_i \in V^2, D^2 \setminus \{(j, j) \mid j \in [k]\})$ is the constraint corresponding to i^{th} edge $\mathbf{e}$ for all $i \in [|E(G)|]$. The constraint states that any vertex pair connected by an edge must not have the same colour.

The solution set to this CSP will be the set of colourings of G for which no edge has vertex endpoints of the same colour.

Example 2.2. 3-Sat of a Boolean formula Φ with n clauses, can be interpreted as the triple $(X, T, \{C_1, C_2, ..., C_n\})$, where:

- $X = \{x_1, x_2, x_3\}$ is the set of literals,
- $T = \{\text{TRUE}, \text{FALSE}\}$ is the set of truth assignments, and
- $C_i = ((x_1, x_2, x_3)_i, T^3 \setminus (\text{FALSE}, \text{FALSE}, \text{FALSE}))$ is the constraint corresponding to the i^{th} clause in Φ. The constraint states that, for all $i \in [n]$, the clause $(x_1, x_2, x_3)_i$, must have at least one true literal.

A solution to this CSP will be the set of truth assignments for Φ such that at least one literal is true in every clause.

Example 2.3. A system of linear equations over the set $\mathbb{Z}_k = \{0, 1, ..., k\}$ can be interpreted as the triple $(X, \mathbb{Z}_k, \{C_1, C_2, ..., C_m\})$. Let $A_{m\times n}$ be the matrix, $\mathbf{b}_{m\times 1}$ be the column vector, and $\mathbf{x}_{m\times 1}$ be the variable vector defining our linear system $A\mathbf{x} = \mathbf{b}$, where:

- $X = \{x_1, x_2, ..., x_n\}$ are the entries of column vector $\mathbf{x} = [x_1 \;\; x_2 \;\; ... \;\; x_n]^\mathsf{T}$,
- $\mathbb{Z}_k = \{0, 1, ..., k\}$ with addition modulo k, and
- $C_i = ((x_1, x_2, ..., x_n), \{(z_1, z_2, ..., z_n) \in \mathbb{Z}_k^n | a_{i,1}z_1 + a_{i,2}z_2 + ... + a_{i,n}z_n = b_i\})$ is the constraint corresponding to the equation in the i^{th} row for all $i \in [m]$. The constraint states that the left-hand side of the equation must equal the right-hand side of the equation for every row.

The input size of a CSP instance $\mathcal{P} = (V, A, \mathcal{C})$ is $n = |V|$. By definition, there are $|A|^n$ possible mappings from V to A which puts some algorithms which solve CSPs into an exponential time complexity class much harder than the polynomial time classes **P** and **NP** [**13**]. In order to find the complexity of verifying a CSP, it is helpful to interpret CSPs using an alternative definition.

2.2 CSP as a Homomorphism Problem

CSPs can be interpreted as the problem of finding homomorphisms for relational structures. A *relation* R on a set A is a subset $R \subseteq A^n$ for some fixed positive integer n called the *arity* of R.

Definition 2.2. A *relational structure* is a pair $\mathbb{A} = (A, R^{\mathbb{A}})$, where A is a set and $R^{\mathbb{A}}$ is a relation on A. In more obvious cases, we will write R instead of $R^{\mathbb{A}}$ for relations which pertain to a relational structure $\mathbb{A}$. Let $\mathcal{R}$ be a finite set of relation symbols with their associated arities; we will call $\mathcal{R}$ a *similarity type.* A relational structure $\mathbb{A} = (A, R)$ is of *type* $\mathcal{R}$ if $R \subseteq \mathcal{R}$.

For example, a graph or digraph is a relational structure $G = (V, E)$, where $E \subseteq V^2$. A pair of relational structures $\mathbb{A}$ and $\mathbb{B}$ are called *similar* if they have the same similarity type. Like with graphs, a homomorphism from one relation to another can be defined.

Definition 2.3. Let $\mathbb{A} = (A, R^{\mathbb{A}})$ and $\mathbb{B} = (B, R^{\mathbb{B}})$ be similar relational structures. A *homomorphism* from $\mathbb{A}$ to $\mathbb{B}$ is a function $h : A \longrightarrow B$ such that if $r \in R^{\mathbb{A}}$, then $h(r) \in R^{\mathbb{B}}$. We write $\mathbb{A} \longrightarrow \mathbb{B}$ if there exists at least one homomorphism from $\mathbb{A}$ to $\mathbb{B}$.

For a relational structure $\mathbb{B} = (B, R)$, if we take B to be a set of variables (a scope) and R to be a set of constraints (a template), we can take the CSP of $\mathbb{B}$. By definition, the solution set of the CSP of $\mathbb{B}$ is the set

$$\mathrm{CSP}(\mathbb{B}) = \{\mathbb{A} \mid \mathbb{A} \longrightarrow \mathbb{B}\}.$$

Example 2.4. The k-Colourability problem can be expressed as $\mathrm{CSP}(\mathbb{T})$, where $\mathbb{T} = (\{1, 2, ..., k\}, \{1, 2, ..., k\}^2 \setminus \{(i, i) \mid i \in [k]\})$ is the complete graph K_k. Graphs which are homomorphic to $\mathbb{T}$ are k-colourable by colouring all of the vertices which are mapped to each element in $\{1, 2, ..., k\}$ the same colour.

A homomorphism h from $\mathbb{A} = (A, R^{\mathbb{A}})$ to $\mathbb{B} = (B, R^{\mathbb{B}})$ is a mapping of size n, where $n = |A|$. Verifying that h is a homomorphism can be done in polynomial time with respect to n since we must check that for all $r \in R^{\mathbb{A}}$, we have $h(r) \in R^{\mathbb{B}}$ given that the size of $R^{\mathbb{A}}$ is not large. Since the size of $R^{\mathbb{A}}$ is polynomial in n, we can see that the problem of verifying a CSP solution belongs in **NP**; more specifically, CSPs are known to be **NP**-complete [**7**]. But, it is possible to find tractable cases of CSP once we apply restrictions on the constraint templates.

2.3 Restricted CSPs

In this section, we will introduce a subclass of CSPs which have restrictions on their constraint templates.

Definition 2.4. Let $\mathcal{P} = (V, A, \{C_1, C_2, ..., C_q\})$ be a CSP instance with constraints $C_i = (\mathbf{s}_i, R_i)$ and let Γ be a set of relations on A. We denote the subclass CSP(Γ) as being the class of CSPs such that $R_i \in \Gamma$ for all $i \in [q]$. Every constraint template is restricted to the relations in Γ. We call Γ the *constraint language* of $\mathcal{P}$.

Example 2.5. In reference to Example 2.1, we consider the constraint template $R = D^2 \setminus \{(i, i) \mid i \in [k]\}$ from the constraints for k-Colourability. The subclass CSP(R) includes the CSP for k-Colourability.

By conforming to specific conditions, it is possible to create a constraint language Γ which yields a tractable subclass CSP(Γ). A constraint language Γ (not necessarily finite) is called *locally tractable* if for every finite subset $\Gamma' \subseteq \Gamma$, the class CSP(Γ') is tractable, and Γ is called *globally tractable* if CSP(Γ) itself is tractable. Since we are dealing with finite constraint languages in this paper, we will refer to both locally and globally tractable constraint languages as simply being tractable. Otherwise, Γ is called **NP**-complete if CSP(Γ) is. The Dichotomy Conjecture introduced in [**11**] states that CSP(Γ) is either tractable or **NP**-complete for all constraint languages Γ. The conjecture was proven for constraint languages of size 2 [**17**], and for those of size 3 [**6**] among others;

however, it remains open for all constraint languages of size greater than 3. In order to determine which constraint languages are tractable, it is useful to study their algebraic properties.

2.4 Algebras

While the study of universal algebra is a distinct field of mathematics on its own, mounting evidence has shown that the algebraic properties of CSPs have a significant impact on their complexity [**1**]. Before we proceed, we must define an algebraic structure (or just algebra) as a mathematical object.

Definition 2.5 ([**9**])**.** An *algebra* $\mathsf{A} = (A, \mathcal{F})$ is a pair where A is a set of elements called the *universe* of A, and $\mathcal{F}$ is a set of *operations* (or functions) which can be applied to the elements in A, and are closed under A. The arity of an operation $f \in \mathcal{F}$ is the number of elements the operation accepts as its argument. An operation is called k-ary if it has an arity of k. A *term operation* in A is an operation which can be obtained by repeated compositions of the operations in $\mathcal{F}$.

Example 2.6. The pair $\mathsf{A} = (\mathbb{Z}, \{+, \times\})$ is the algebra on which integer arithmetic is done. Our universe is the set of integers $\mathbb{Z}$, and our operations are both the binary sum and product operations. This particular algebra is called the *ring of integers*.

Definition 2.6. An algebra $\mathsf{B} = (B, \mathcal{F})$ is a *subalgebra* of $\mathsf{A} = (A, \mathcal{F})$ if $B \subseteq A$ and B is closed under the operations in $\mathcal{F}$. We denote this as $\mathsf{B} \leq \mathsf{A}$. If A is the universe of an algebra $\mathsf{A} = (A, \mathcal{F})$ and the subset $B \subseteq A$ is closed under the operations in $\mathcal{F}$, we call B a *subuniverse* of A (or a subuniverse of A itself).

Example 2.7. The algebra $\mathsf{B} = (2\mathbb{Z}, \{+, \times\})$ is a subalgebra of the ring of integers $\mathsf{A} = (\mathbb{Z}, \{+, \times\})$ presented in Example 2.6. Taking $2\mathbb{Z}$ to be the set of even integers, it is clear that $2\mathbb{Z} \subseteq \mathbb{Z}$ and that $2\mathbb{Z}$ is closed under $\{+, \times\}$ since taking the sum or product of two even integers always yields an even integer.

Definition 2.7. The *direct product* of a pair of algebras A and B which share the same set of operations $\mathcal{F}$ is the algebra $\mathsf{C} = (A \times B, \mathcal{F})$, where $A \times B$ denotes the Cartesian product $\{(a, b) \mid a \in A, b \in B\}$. The direct product of algebras A and B is denoted $\mathsf{A} \times \mathsf{B}$. The elements in $A \times B$ are still closed under the operations in $\mathcal{F}$ which are applied element-wise to each component a and b in a tuple $(a, b) \in A \times B$. We will use the notation A^n to denote the direct product of n copies of an algebra A.

Algebras can be classified based on properties corresponding to their universes and operations. Here, we will define some of the properties associated with algebras.

Definition 2.8. Let $\mathsf{A} = (A, \mathcal{F})$ be an algebra and $*$ be a binary operation which accepts, as arguments, all elements from the universe A. The following are referred to as algebraic axioms:

- *Associativity*: For all $x, y, z \in A$, we have $(x * y) * z = x * (y * z)$.
- *Commutativity*: For all $x, y \in A$, we have $x * y = y * x$.

- *Closure*: For all $x, y \in A$, we have $(x * y) \in A$.
- *Identity element*: For all $x \in A$, there exists some $i \in A$ such $x * i = x$.
- *Inverse element*: For all $x \in A$ and some identity element $i \in A$, there exists some $x^{-1} \in A$ such that $x * x^{-1} = i$.

Different classes of algebras exist depending on which axioms are obeyed and by which operations. Here, we present some of the different classes of algebras which will be relevant to our paper.

Definition 2.9. A *field* $\mathsf{F} = (F, \{+, \times\})$ is an algebra which consists of a set of elements and the addition and product operations. The operations $+$ and $\times$ must satisfy the following field axioms for all $x, y, z \in F$:

- Associativity: We have $(x+y)+z = x+(y+z)$ and $(x\times y)\times z = x\times(y\times z)$.
- Commutativity: We have $x + y = y + x$ and $x \times y = y \times x$.
- Distributivity: We have $x \times (y + z) = (x \times y) + (x \times z)$.
- Identity elements: There exists an element $i \in F$ such that $x + i = x$ and there an element $i' \in F$ such that $x \times i' = x$.
- Inverse elements: For an additive identity element i, there exists some $-x \in F$ such that $x + (-x) = i$. For a multiplicative identity element i', there exists some $x^{-1} \in F$ such that $x \times x^{-1} = i'$

Definition 2.10. A *group* $\mathsf{G} = (G, *)$ is an algebra which consists of a set of elements and a single operation $*$. The operation must satisfy the following group axioms for all $x, y, z \in G$:

- Associativity: We have $(x * y) * z = x * (y * z)$.
- Closure: We have $x * y \in G$.
- Identity elements: There exists an element $i \in G$ such that $x * i = x$.
- Inverse elements: For some identity element i, there exists some $x^{-1} \in G$ such that $x * x^{-1} = i$.
- Commutativity (for *Abelian* groups only): We have $x * y = y * x$.

Definition 2.11. A *partially ordered set* (or *poset*) $P = (X, \preceq)$ is a set X and an ordering $\preceq$, which is transitive, antisymmetric, and reflexive. The *supremum* (or *join*) of X is an element $\sup(X) = s \in X$ such that for all $x \in X$, we have $x \preceq s$, and the *infimum* (or *meet*) of X is an element $\inf(X) = i \in X$ such that for all $x \in X$, we have $i \preceq x$. These properties essentially arrange the elements in X from "smallest" to "greatest".

Definition 2.12. A *lattice* $\mathsf{L} = (L, \{\wedge, \vee\})$ is an algebra which consists of a poset L, the meet operation $\wedge$, and the join operation $\vee$. The operations $\wedge$ and $\vee$ must satisfy the following lattice axioms for all $x, y, z \in L$:

- Associativity: We have $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ and $(x \vee y) \vee z = x \vee (y \vee z)$.
- Commutativity: We have $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$.
- Absorption: If $x \preceq y$, we have $x \wedge (x \vee y) = x \vee (x \wedge y) = x$.
- Idempotency: We have $x \wedge x = x \vee x = x$.

Now that we have studied the different algebraic classes we will see, we will study the notion of algebraic homomorphisms.

Definition 2.13. Let $\mathsf{A} = (A, \mathcal{F}_\mathsf{A})$ and $\mathsf{B} = (B, \mathcal{F}_\mathsf{B})$ be algebras. A function $h : A \longrightarrow B$ is a called *homomorphism* from A to B if

$$h(f_\mathsf{A}(a_1, a_2, ..., a_n)) = f_\mathsf{B}(h(b_1), h(b_2), ..., h(b_n))$$

for all n-ary operations $f_\mathsf{A} \in \mathcal{F}_\mathsf{A}$ and $f_\mathsf{B} \in \mathcal{F}_\mathsf{B}$. A homomorphism h from A to B is called

(1) an *isomorphism* if the inverse h^{-1} is a homomorphism from B to A,

(2) an *endomorphism* if $\mathsf{A} = \mathsf{B}$, or

(3) an *automorphism* if it is both an isomorphism and an endomorphism.

Example 2.8. Let $\mathsf{G} = (\mathbb{R}, +)$ be the group under addition with the universe of real numbers and let $\mathsf{H} = (\mathbb{R}^+, \times)$ be the group under multiplication with the universe of positive real numbers. Let $h : \mathbb{R} \longrightarrow \mathbb{R}^+$ be the function $h(x) = \mathsf{e}^x$, where e is the exponential constant. We can see that h is a homomorphism from G to H since for all $x, y \in \mathbb{R}$, we have

$$h(x + y) = \mathsf{e}^{x+y} = \mathsf{e}^x \times \mathsf{e}^y = h(x) \times h(y).$$

The relations in a constraint language alone do not meet the definition of an algebra. But, by taking the domain of a CSP along with the relations in the constraint language, it is possible to create operations accepted by the domain which preserve the relations.

2.5 Polymorphisms

In order to study the algebraic properties of Γ, we must create a set of operations using the relations in Γ.

Definition 2.14. Let $R \subseteq A^k$ be a k-ary relation on A and let $\phi : A^n \longrightarrow A$ be an n-ary operation on A. ϕ is called a *polymorphism* of R (or R is *invariant under* ϕ) if for all n-tuples $(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^n)$, where $\mathbf{r}^i = (r_1^i, r_2^i, ..., r_k^i) \in R$, we have

$$\phi(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^n) = (\phi(r_1^1, r_1^2, ..., r_1^n), \phi(r_2^1, r_2^2, ..., r_2^n), ..., \phi(r_k^1, r_k^2, ..., r_k^n)) \subseteq R.$$

If Γ is a set of relations on A, then

$$\mathrm{Pol}(\Gamma) = \{\phi \mid \phi \text{ is a polymorphism of all } R \in \Gamma\}.$$

If Φ is a set of operations on A, then

$$\mathrm{Inv}(\Phi) = \{R \mid R \text{ is invariant under all } \phi \in \Phi\}.$$

Example 2.9. Let R be a relation denoting the solution set of a system of linear equations on a finite field. More specifically, for some matrix A and column vector $\mathbf{b}$ defining our linear system, $R = \{\mathbf{r} \mid A\mathbf{r} = \mathbf{b}\}$. Let

$\phi : R^3 \longrightarrow R$ be the operation $\phi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbf{x} - \mathbf{y} + \mathbf{z}$. We can see that ϕ is a polymorphism of R. Indeed, ϕ preserves the relation since for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$, we have,

$$A\phi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A(\mathbf{x} - \mathbf{y} + \mathbf{z}) = A\mathbf{x} - A\mathbf{y} + A\mathbf{z} = \mathbf{b} - \mathbf{b} + \mathbf{b} = \mathbf{b}.$$

Given a CSP instance $\mathcal{P} = (V, A, \mathcal{C})$ with constraint language Γ, we can create an algebra $\mathsf{A}_\Gamma = (A, \text{Pol}(\Gamma))$. Conversely, given an Algebra $\mathsf{B} = (B, \mathcal{F})$, we can create a constraint language $\Gamma_\mathsf{B} = \text{Inv}(\mathcal{F})$. Just as with constraint languages, we call an algebra B tractable or **NP**-complete if the constraint language Γ_B is. Through past research, it has been shown that the operations in Pol(Γ) play a significant role in the complexity of the subclass CSP(Γ) [**1**].

2.6 The Mal'tsev Operation

In [**7**], it was shown by Bulatov and Dalmau that a subclass CSP(Γ) is tractable if Pol(Γ) contained a Mal'tsev operation.

Definition 2.15. A *Mal'tsev* operation on a set A is a function $m : A^3 \longrightarrow A$ such that for all $x, y \in A$, we have $m(x, y, y) = m(y, y, x) = x$.

Given a Mal'tsev operation m, we can take the relations invariant to m to create a tractable subclass CSP(Inv(m)), where Inv(m) is the set of constraint languages which have the operation m in their polymorphisms.

Example 2.10. In Example 2.9, we mentioned that the operation $\phi(x, y, z)$

was a polymorphism of the solution set, R, of a system of linear equations over a finite field. ϕ also happens to be a Mal'tsev operation on the solution set since for all $x, y \in R$, we have

$$\phi(x, y, y) = x - y + y = x \quad \text{and} \quad \phi(y, y, x) = y - y + x = x.$$

Since R is an invariant under ϕ, the problem of solving a system of linear equations on a finite field is tractable.

Since the set of Mal'tsev operations only includes those with arity 3, the related CSP subclass does not fully encapsulate the entire subclass of tractable CSPs. Over the years, efforts have been made to expand on the known tractable subclasses and many results have been obtained. Dalmau himself was able to improve on his result involving Mal'tsev operations by showing that constraint languages with generalized majority-minority (GMM) operations also yield a tractable subclass [**10**].

Definition 2.16. A *generalized majority-minority* operation on a set A is a function $g : A^k \longrightarrow A$ with $k \geq 3$ such that for all $a, b \in A$, we have

$$g(x, y, y, ..., y) = g(y, x, y, y, ..., y) = g(y, y, ..., y, x) = y, \quad \forall x, y \in \{a, b\},$$

or

$$g(x, y, y, ..., y) = g(y, y, ..., y, x) = x, \quad \forall x, y \in \{a, b\}.$$

A 3-ary GMM operation includes those which are Mal'tsev (see next chapter). Dalmau's GMM result is a generalization of his older result involving

Mal’tsev operations, but it still does not encapsulate the entire subclass of tractable CSPs. In the coming chapters, we will study CSP subclasses whose polymorphisms contain what are called k-edge operations.

Chapter 3

CONSTRAINT LANGUAGES WITH FEW SUBPOWERS

As discussed in the previous chapter, tractable CSP subclasses have been found over the years, but we have yet to fully encapsulate the entire subclass of tractable CSPs. Constraint languages with Mal'tsev operations were shown to be tractable by Bulatov and Dalmau who created an algorithm which can solve any Mal'tsev CSPs in polynomial time [**7**]. This result was later improved by Dalmau himself by showing that CSPs with constraint languages which have GMM operations were also tractable [**10**]; this was done by generalizing his previous algorithm to include operations with higher arity.

In this chapter, we will describe an algorithm which will further improve Dalmau's results by including constraint languages which have what are called k-edge operations. Constraint languages Γ whose polymorphisms contain k-edge operations are tractable [**15**]. This result was presented by Idziak et al and uses a variation of Dalmau's algorithm essentially unchanged. The algorithm presented in this chapter can be seen as a generalization of the algorithm which performs Gaussian elimination on a system of linear equations. The solution sets of CSP(Γ) are known to have generating sets when considered as universes of particular algebras analogous to the basis of a vector space. Furthermore, it is also argued by Idziak et al that constraint languages which do not contain k-edge operations cannot be solved by any variation of their "robust Gaussian-like" algorithm. However, this still leaves open the possibility that constraint languages without k-edge operations can be solved by a different algorithm in polynomial time.

3.1 Preliminaries

Just as GMM operations were a generalization of Mal'tsev operations, k-edge operations are a generalization of both GMM and Mal'tsev operations along with other operations not mentioned.

Definition 3.1. For $k \geq 2$, a *k-edge operation* on a set A is a $(k+1)$-ary operation e which, for all $x, y \in A$, satisfies:

$$\begin{aligned}
e(x,x,y,y,y,...,y,y) &= y \\
e(x,y,x,y,y,...,y,y) &= y \\
e(y,y,y,x,y,...,y,y) &= y \\
e(y,y,y,y,x,...,y,y) &= y \\
&\vdots \\
e(y,y,y,y,y,...,x,y) &= y \\
e(y,y,y,y,y,...,y,x) &= y.
\end{aligned}$$

We note the "shepherd's crook" **[15]** shape formed by the occurrence of the variable x in the previous definition. It is clear that an operation is a 2-edge operation if and only if it is a Mal'tsev operation. It was also established in

[5] that if an algebra contains a k-ary GMM operation, then it also contains a k-edge operation. As a result, we can see that a k-edge operation is indeed a generalization of both Mal'tsev and GMM operations. To establish the existence of k-edge operations in the polymorphisms of a constraint language Γ, we must define a type of subalgebra called a subpower.

Definition 3.2. An algebra B is a *subpower* of A if B is a subalgebra of A^n for some positive integer n. We denote this as $\mathsf{B} \leq \mathsf{A}^n$.

Now that we have properly introduced subpowers, we are ready to explain the notion of an algebra having "few" subpowers. In order to do this, we must first define what is called the relational clone of a constraint language.

Definition 3.3. Given a class of relation symbols Ω, a *primitive positive formula* is a formula in the form

$$\exists \alpha_1, \alpha_2, ..., \alpha_n \bigwedge_{i=1}^{n} \phi_i(\diamond, ..., \diamond),$$

where $\diamond$ represents either a free variable or one of variables $\alpha_1, \alpha_2, ..., \alpha_n$, and all formulas ϕ_i use relation symbols from Ω for all $i \in [n]$.

Definition 3.4. Let Γ be a set of relations over a set A. We define a new set of relations $\langle\Gamma\rangle = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$. The set $\langle\Gamma\rangle$ is precisely the set of relations over A definable from Γ using primitive positive formulas where $\Omega = \{=\}$. If $\Gamma = \langle\Gamma\rangle$, then $\langle\Gamma\rangle$ is called the *relational clone* of Γ.

Let Γ be a set of relations over a set A and let $\mathcal{P} = (V, A, \mathcal{C})$ be an instance of CSP(Γ). The solution set of $\mathcal{P}$ can be identified by any n-ary member of $\langle \Gamma \rangle$ (where $n = |V|$) which themselves are the universes of the subpowers of A_Γ. As mentioned earlier in this chapter, the solution sets of CSP(Γ) are known to have generating sets when considered as universes of the subpowers of A_Γ. Dalmau's algorithm requires that the size of these generating sets be "small", meaning that they are of order $\mathcal{O}(n^p)$ for some positive integer p. We introduce some equivalent conditions which ensures that all subpowers of A_Γ have small generating sets.

Definition 3.5. Let A be an algebra and let $\mathcal{S} = \{\mathsf{B} \mid \mathsf{B} \leq \mathsf{A}^n\}$ be the set of all subpowers of A for some positive integer n. Let $s_\mathsf{A}(n) = \log_2|\mathcal{S}|$ and let $g_\mathsf{A}(n)$ be the smallest integer t such that every subpower of A has a generating set of size at most t. For some positive integer p,

(1) if $s_\mathsf{A}(n) = \mathcal{O}(n^p)$, then A has *few subpowers*, and

(2) if $g_\mathsf{A}(n) = \mathcal{O}(n^p)$, then A has *polynomially generated subpowers.*

Theorem 3.1 ([5])**.** Let $\mathcal{P} = (V, A, \mathcal{C})$ be an instance of CSP(Γ) and let $n = |V|$. The following statements are all equivalent for the algebra A_Γ:

(1) A_Γ contains a k-edge operation for some $k \geq 2$.

(2) A_Γ has few subpowers and $s_\mathsf{A}(n) = \mathcal{O}(n^k)$.

(3) A_Γ has polynomially generated subpowers and $g_\mathsf{A}(n) = \mathcal{O}(n^{k-1})$.

The proof for this theorem utilizes three term operations in A_Γ derived from the k-edge operation. The presence of these term operations allows us to present the following lemma.

Lemma 3.1 ([5])**.** Let A be a finite algebra with universe A which contains a k-edge term operation e. Then the algebra (A, e) contains three term operations $d(x, y)$, $p(x, y, z)$, and $s(x_1, x_2, ..., x_k)$ (which are also term operations of A) such that:

$$\begin{aligned} p(x,y,y) &= x \\ p(x,x,y) &= d(x,y) \\ d(x,d(x,y)) &= d(x,y) \\ s(y,x,x,x,x,...,x,x) &= d(x,y) \\ s(x,y,x,x,x,...,x,x) &= x \\ s(x,x,y,x,x,...,x,x) &= x \\ &\vdots \\ s(x,x,x,x,x,...,x,y) &= x. \end{aligned}$$

3.2 Signatures and Representations

In this section, we will focus on the elements in the universe A of an algebra A. For positive integers n and j, let $\mathbf{t} = (t_1, t_2, ..., t_n)$ be an n-ary tuple and let $\mathbf{i} = (i_1, i_2, ..., i_j)$ be a j-ary tuple such that $i_k \in [n]$ for all $k \in [j]$; the elements in $\mathbf{i}$ need not be distinct. In the proceeding definition, we will use these tuples to create a new relation from A.

Definition 3.6. Let $R \subseteq A^n$ be an n-ary relation on A. If $\mathbf{t} \in R$, we define the *projection* of $\mathbf{t}$ on $\mathbf{i}$ as the j-ary tuple $\text{proj}_{\mathbf{i}}\mathbf{t} = \text{proj}_{i_1,i_2,...,i_j}\mathbf{t} = (t_{i_1}, t_{i_2}, ..., t_{i_j})$. Similarly, we can take the projection of R itself on $\mathbf{i}$ which creates the j-ary relation $\text{proj}_{\mathbf{i}}R = \{\text{proj}_{\mathbf{i}}\mathbf{t} \mid \mathbf{t} \in R\}$.

Referring to the previous section, consider an algebra A which contains a k-edge operation e and the term operations d, p, and s as defined in Lemma 3.1. The pair $(a, b) \in A^2$ is called a *minority pair* if $d(a, b) = b$. We call the triple $\mathbf{I} = (i, a, b) \in [n] \times A^2$ an *index of rank* n if $i \in [n]$ and $a, b \in A$ and we call $\mathbf{I}$ a *minority index* if (a, b) is a minority pair.

Definition 3.7. Let A be a finite set, $\mathbf{t}_1$ and $\mathbf{t}_2$ be n-ary tuples, and let the tuple $(i, a, b) \in [n] \times A^2$ be a minority index. We say that the pair $(\mathbf{t}_1, \mathbf{t}_2)$ *witnesses* (i, a, b) if $\text{proj}_{1,2,...,i-1}\mathbf{t}_1 = \text{proj}_{1,2,...,i-1}\mathbf{t}_2$, $\text{proj}_i\mathbf{t}_1 = a$, and $\text{proj}_i\mathbf{t}_2 = b$; to be clear, the first $i - 1$ elements in $\mathbf{t}_1$ and $\mathbf{t}_2$ match while their i^{th} elements are equal to a and b respectively. Let $R \subseteq A^n$ be a relation. We define the *signature* of R to be the set of tuples

$$\text{Sig}_R = \{(i, a, b) \in [n] \times A^2 \mid \exists \mathbf{t}_1, \mathbf{t}_2 \in R \text{ witnesses } (i, a, b)\}.$$

The size of the signature of a relation R depends on how closely pair of tuples in R match each other. Let $S \subseteq R$ be a relation. We call S a *representation* of R if $\text{Sig}_S = \text{Sig}_R$. Furthermore, we call S a *compact* representation of R if $|S| \leq 2|\text{Sig}_R|$. In fact, it was proven by Dalmau [**10**] that it is possible to construct a compact representation of any relation on a finite set.

Example 3.1. Let $A = \{1, 2, 3, 4\}$ and let

$$R = \{(1, 4, 1, 3), (1, 4, 1, 2), (2, 2, 1, 3), (2, 2, 4, 4), (3, 4, 3, 3)\} \subseteq A^4.$$

We can see that $\mathrm{Sig}_R = \{(3, 3, 2), (2, 1, 4)\}$. A representation of R is

$$S = \{(1, 4, 1, 3), (1, 4, 1, 2), (2, 2, 1, 3), (2, 2, 4, 4)\} = R \setminus (3, 4, 3, 3)$$

since $S \subseteq R$ and $\mathrm{Sig}_S = \mathrm{Sig}_R$. We can also see that S is a compact representation of R since $4 = |S| \leq 2|\mathrm{Sig}_R| = 2(2) = 4$.

Let R be an n-ary relation on A and let ϕ be an operation. We define the new relation $\langle R \rangle_\phi$ to be the smallest n-ary relation which contains R and is invariant under ϕ; we call $\langle R \rangle_\phi$ the *closure* of R under ϕ. It is clear from the definition that $R \subseteq \langle R \rangle_\phi$.

Theorem 3.2 ([**5**])**.** Let A be a finite algebra with a k-edge operation e for some $k \geq 2$ and term operations d, p, and s as defined in Lemma 3.1, let B be a subpower of A with universe B, and let R be representation of B. Suppose $\mathbf{I} = (i, a, b)$ is a minority index witnessed in B. For all $\mathbf{t} \in \langle R \rangle_e$, where $\mathrm{proj}_i = a$, there exists $\mathbf{u} \in \langle R \rangle_e$ such that $(\mathbf{t}, \mathbf{u})$ witness $\mathbf{I}$.

Going back to compact representations, if A is an algebra with universe A with a k-edge operation and B is a subpower with universe B, then every subset $B \subseteq A^n$ will have a representation with size bounded above by a polynomial in n of degree $k-1$. More specifically, if we set $m = \min\{k-1, n\}$, every subset $B \subseteq A^n$ will have a representation of size at most

$$2|\mathrm{Sig}_B| + \sum_{\substack{T \in [n] \\ |T| \leq m}} |\mathrm{proj}_T B| = \mathcal{O}(n^{k-1}).$$

This fact stems from the claim in Theorem 3.1 that the function $g_{\mathsf{A}}(n)$ is of order $\mathcal{O}(n^{k-1})$ when an algebra A contains a k-edge operation.

3.3 A Tractability Algorithm

With some slight modifications to the algorithm presented in **[10]** which solves CSPs with GMM operations, the following theorem was proven.

Theorem 3.3 ([**15**]). Let A be a finite algebra with universe A with few subpowers, and let Γ be the constraint language consisting of all subuniverses of A^n for some positive integer n. Then the class CSP(Γ) is tractable.

To prove that CSPs with few subpowers were tractable, it sufficed to prove that CSPs with k-edge operations were tractable since both were equivalent. For the duration of this chapter, we will set $\mathsf{A} = (A, \phi(x_1, x_2, ..., x_k))$ to be an algebra with a k-edge operation ϕ. Let $\mathcal{P} = (V, A, \mathcal{C})$ be an instance of CSP(Γ_{A}), where $V = \{v_1, v_2, ..., v_n\}$ is the set of variables, A is the universe of A, and $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ is the set of constraints. For the purposes of the upcoming algorithm, we will use $\mathcal{P}_l = (V, A, \mathcal{C}_l)$ to denote the CSP where C_l contains the first l constraints of C for some $l \leq m$. We will set R and R_l to denote the solution set of $\mathcal{P}$ and $\mathcal{P}_l$ respectively.

The algorithm presented by Idziak et al begins with the compact representation of R_0 (equal to A^n). It will then recursively add in each constraint C_l producing a solution set R_l for $0 \leq l \leq m$ until we are left with $R_m = R$, a compact representation of the solution set of $\mathcal{P}$. It is clear that $R_m \subseteq R_{m-1} \subseteq ... \subseteq R_0$ since the addition of each constraint C_l will eliminate possible solutions which had only satisfied the constraints before it. If R_m is empty, then there is no solution to the CSP. The following is a pseudocode describing Dalmau's algorithm.

```
Algorithm Dalmau$((\{v_1, v_2, ..., v_n\}, A, \{C_1, C_2, ..., C_m\}))$
Step 1      set $R'_0$ some compact representation of $A^n$
Step 2      for each $l \in \{0, 1, ..., m-1\}$ do
              (let $C_{l+1}$ be $((v_{i_1}, v_{i_2}, ..., v_{i_{k_{l+1}}}), S_{l+1})$)
Step 2.1      set $R'_{l+1} :=$ Next$(R'_l, i_1, i_2, ..., i_{k_{l+1}}, S_{l+1})$
            end for each
Step 3      if $R'_m \neq \emptyset$ return "yes"
Step 4      return "no"
```

The subroutine `Next` uses the subroutine `Next-Beta` which itself uses subroutines `Nonempty` and `Fix-Values` all of which are fully described and analyzed in [**15**]. Here, we provide a brief description of each subroutine.

`Nonempty`$(R', i_1, i_2, ..., i_j, S)$ receives, as input, a compact representation R' of a subuniverse $R \subseteq A^n$, a sequence of elements $i_1, i_2, ..., i_j$ from $[n]$, and a subuniverse $S \subseteq A^n$. If there exists some tuple $\mathbf{t} \in R$ such that $\text{proj}_{i_1,i_2,\ldots,i_j}\mathbf{t} \in S$, then `Nonempty` outputs $\mathbf{t}$; otherwise, it will output **"no"**. The running time of `Nonempty` is bounded above by $\mathcal{O}(((n|A|)^k + |\text{proj}_{i_1,i_2,\ldots,i_j}R|)^{k+2}n|S|)$ [**15**]. Although the size of $\text{proj}_{i_1,i_2,\ldots,i_j}R$ may be exponential with regards to the size of the input n, Dalmau was able to deal with this issue by invoking `Nonempty` in such a way as to bound $|\text{proj}_{i_1,i_2,\ldots,i_j}R|$ above polynomially [**10**]. This version of `Nonempty` is unchanged from the one used in [**10**].

```
Algorithm Nonempty($R', i_1, i_2, ..., i_j, S$)
1       set $U := R'$
2       while $\exists \mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_k \in U$ such that
        $\text{proj}_{i_1,i_2,\ldots,i_j}\phi(\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_k) \notin \text{proj}_{i_1,i_2,\ldots,i_j}U$ do
2.1         set $U := U \cup \{\phi(\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_k)\}$
        endwhile
3       if $\exists \mathbf{t}$ in $U$ such that $\text{proj}_{i_1,i_2,\ldots,i_j}\mathbf{t} \in S$ then return $\mathbf{t}$
4       else return "no"
```

`Fix-Values`$(R', a_1, a_2, ..., a_m)$ receives, as input, a compact representation R' of a subuniverse $R \subseteq A^n$, and a sequence of elements $a_1, a_2, ..., a_m$ from A. It outputs a compact representation

$$\{\mathbf{t} \in R : \text{proj}_{[m]}\mathbf{t} = (a_1, a_2, ..., a_m)\} \subseteq A.$$

The running time of `Fix-Values` is bounded above by $\mathcal{O}((n|A|)^{(k+1)(k+2)})$ [**15**]. This version of `Fix-Values` differed slightly from that in [**10**]; *Step 2.2.1.2* was deleted and *Step 2.2.1.1* was changed to "**set** $\mathbf{t}_5 := p(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$", where $p(x, y, z)$ is the 3-ary term operation as defined in Lemma 3.1.

```
Algorithm Fix-Values($R', a_1, a_2, ..., a_m$)
1          set $j := 0; U_j := R'$
2          while $j < m$ do
2.1          set $U_{j+1} := \emptyset$
2.2          for each $(i, a, b) \in [n] \times A^2$, with $(a, b)$ a minority pair, do
2.2.1          if Nonempty($U_j, j+1, i, \{(a_{j+1}, a)\}$) $\neq$ "no" and
                 $\exists \mathbf{t}_2, \mathbf{t}_3 \in U_j$ witnessing $(i, a, b)$ and
                 $i > j+1$ then
                   (let $\mathbf{t}_1$ be the tuple returned by
                   Nonempty($U_j, j+1, i, \{(a_{j+1}, a)\}$))
2.2.1.1            set $\mathbf{t}_5 := p(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$
2.2.1.2            set $U_{j+1} := U_{j+1} \cup \{\mathbf{t}_1, \mathbf{t}_5\}$
             end for each
2.3          for each $k' \in [k-1]$
               for each $l_1, l_2, ..., l_{k'} \in [n]$ with $l_1 < l_2 < ... < l_{k'}$
                 for each $d_1, d_2, ..., d_{k'} \in A$ do
2.3.1              if Nonempty($U_j, l_1, l_2, ..., l_{k'}, j+1, \{(d_1, d_2, ..., d_{k'}, a_{j+1})\}$)
                   $\neq$ "no"
                   then (let $\mathbf{t}_6$ be the tuple returned by
                   Nonempty($U_j, l_1, l_2, ..., l_{k'}, j+1, \{(d_1, d_2, ..., d_{k'}, a_{j+1})\}$))
                   set $U_{j+1} := U_{j+1} \cup \{\mathbf{t}_6\}$
             end for each
```

```
2.4         set $j := j + 1$
        end while
3       return $U_m$
```

`Next`$(R', i_1, i_2, ..., i_j, S)$ receives, as input, a compact representation R' of a subuniverse $R \subseteq A^n$, a sequence of elements $i_1, i_2, ..., i_j$ from $[n]$, and a subuniverse $S \subseteq A^j$. It outputs a compact representation of the subuniverse

$$R^* = \{\mathbf{t} \in R : \mathrm{proj}_{i_1,i_2,...,i_j}\mathbf{t} \in S\} \subseteq A.$$

`Next` uses a subroutine `Next-Beta` similar to itself, but has a potentially worse running time. The running time of `Next` was shown to be bounded above by $\mathcal{O}((n|A||S|)^{(k+2)^2})$ [**15**]. This version of `Next` is unchanged from the one used in [**10**].

Algorithm `Next-Beta`$(R', i_1, i_2, ..., i_j, S)$

```
1       set $U := \emptyset$
2       for each $(i, a, b) \in [n] \times A^2, \{a, b\}$ is a minority pair do
2.1         if Nonempty$(R', i_1, i_2, ..., i_j, i, S \times \{a\}) \neq$ "no"
            then
              (let $\mathbf{t}_1$ be Nonempty$(R', i_1, i_2, ..., i_j, i, S \times \{a\})$)
2.2           if Nonempty(
              Fix-Values$(R', \mathrm{proj}_1\mathbf{t}, \mathrm{proj}_2\mathbf{t}, ..., \mathrm{proj}_{i-1})$,
              $\mathbf{t}, i_1, i_2, ..., i_j, i, S \times \{b\})) \neq$ "no"
                (let $\mathbf{t}_2$ be Nonempty(
                Fix-Values($R', \mathrm{proj}_1\mathbf{t}, \mathrm{proj}_2\mathbf{t}, ..., \mathrm{proj}_{i-1}$),
                $\mathbf{t}, i_1, i_2, ..., i_j, i, S \times \{b\}$))
                set $U := U \cup \{\mathbf{t}_1, \mathbf{t}_2\}$
        end for each
3       for each $k' \in [k-1]$
            for each $l_1, l_2, ..., l_{k'} \in [n]$ with $l_1 < l_2 < ... < l_{k'}$
              for each $d_1, d_2, ..., d_{k'} \in A$ do
3.1           if Nonempty($R, i_1, i_2, ..., i_j, l_1, l_2, ..., l_{k'}, S \times \{(d_1, d_2, ..., d_{k'})\}$)
              $\neq$ "no"
              then
                (let $\mathbf{t}_3$ = Nonempty($R, i_1, i_2, ..., i_j, l_1, l_2, ..., l_{k'}$,
                $S \times \{(d_1, d_2, ..., d_{k'})\}$))
                set $U := U \cup \{\mathbf{t}_3\}$
4       return $U$
```

Theorem 3.4 ([**15**]). The algorithm `Dalmau` determines whether an instance $\mathcal{P}$ of CSP(Γ_{A}) has a solution in $\mathcal{O}(m(n|A||S^*|)^{k+2^2})$-many steps where n is the number of variables, m is the number of constraints, and S^* is the largest constraint relation to appear in $\mathcal{P}$.

The algorithm presented in [**15**] is the most general tractable "robust Gaussian-like" algorithm there is for solving CSPs. Both Gaussian elimination on a system of linear equations over a finite field and Dalmau's algorithm in [**10**] are systems of constrains with n variables and the solution sets each have generating sets of size polynomial in n with respect to an appropriate algebraic structure. Furthermore, Dalmau's algorithm applies globally to the class $\mathrm{CSP}(\langle\Gamma\rangle)$ where $\langle\Gamma\rangle = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$.

Chapter 4

REDUCTION AND ANALYSIS OF OUR RELATIONAL STRUCTURE

In this chapter, we will design and describe an algorithm which can solve a CSP with few subpowers using a Boolean circuit. Before we can start solving a given CSP, we must reduce our instance to one which fits our algorithm. In this case, we desire to reduce the relational structure associated with the CSP to one which contains only binary relations. Afterwards, we will alter our CSP by implementing a consistency checking algorithm designed to use existing constraints to create new ones of arity 2 and 3 helping us narrow down our solution set. Once we have a fully reduced CSP, we will study the algebraic properties of its relational structure. More specifically, we will study the different types of subalgebras of the algebra created using its relational structure.

4.1 Reduction to a Binary Relational Structure

By interpreting CSPs as homomorphism problems between relational structures, Hell and Něsetřil proved the Dichotomy conjecture for simple graphs [**14**] and Barto et al improved the result by including finite digraphs with no sources or sinks [**2**]. Feder and Vardi were able to reduce, in polynomial time, the problem of proving the Dichotomy Conjecture to the homomorphism problem for digraphs whose vertices can be partitioned into levels (balanced digraphs) [**11**]. More specifically, it was shown that any relational structure $\mathbb{A}$ can be reduced to a balanced digraph $\mathcal{D}(\mathbb{A})$ such that CSP($\mathbb{A}$) is polynomial-time equivalent to CSP($\mathcal{D}(\mathbb{A})$).

Just as with constraint languages in the form of relations, we can create an algebra using the polymorphisms of the relations in a relational structure. Let $\mathbb{A} = (A, R)$ be a relational structure. We define the *algebra of polymorphisms* $\mathsf{A}_{\mathbb{A}}$ to be the algebra $\mathrm{Pol}(\mathbb{A}) = (A, \mathrm{Pol}(R))$. Just as with constraint languages, we call a relational structure $\mathbb{A}$ tractable or **NP**-complete if CSP($\mathbb{A}$) is. For relational structures whose algebra of polymorphisms has few subpowers, the complexity of the reduction itself from $\mathbb{A}$ to $\mathcal{D}(\mathbb{A})$ was improved in [**8**] who showed that it was possible to perform the reduction in logspace. Before we describe this reduction, we must first define a new type of relational structure (or constraint language) which is equivalent to one which has few subpowers.

Definition 4.1. An algebra is called *congruence modular* if there exist operations $m_0(x, y, z, u), m_1(x, y, z, u), ..., m_n(x, y, z, u)$ satisfying

$$\begin{aligned}
&m_0(x, y, z, u) = x \\
&m_i(x, y, y, x) = x, && \text{for all } i \leq n \\
&m_i(x, x, y, y) = m_{i+1}(x, x, y, y) && \text{for all even } i \leq n \\
&m_i(x, y, y, z) = m_{i+1}(x, y, y, z) && \text{for all odd } i \leq n \\
&m_n(x, y, z, u) = u
\end{aligned}$$

The reduction we will describe reduces congruence modular relational structures (among others) to a balanced digraph in logspace. It was also proven in **[19]** that congruence modular relational structures are equivalent to those which have few subpowers. The following theorem was established.

Theorem 4.1 ([**8**])**.** Let $\mathbb{A}$ be a finite relational structure and let $\mathcal{D}(\mathbb{A})$ be the associated digraph. On the algebra of polymorphisms of $\mathbb{A}$ and $\mathcal{D}(\mathbb{A})$, there exist congruence modular operations if and only if there exists a k-edge operation for some $k \geq 4$ (equivalent to having few subpowers).

For the construction of $\mathcal{D}(\mathbb{A})$ in [**8**], vertices were connected by what were called *oriented paths.* An oriented path is a sequence of vertices $v_1, v_2, ..., v_k$ such that for every $i \in [k]$, only one of the pairs, (v_i, v_{i+1}) and (v_{i+1}, v_i), contains a directed edge between them. These oriented paths were made up of two kinds of components made up of vertices $\bullet$ and directed edges $\bullet \longrightarrow \bullet$. A *single edge* between two vertices was simply the directed edge $\bullet \longrightarrow \bullet$ and a *zigzag* between four vertices was the oriented path $\bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet$. Each of these components were partitioned in a manner such that each would add one level to the balanced digraph (hence the name "zigzag"). The oriented path connecting each vertex in $\mathcal{D}(\mathbb{A})$ was made up of a concatenation of single edges and zigzags (denoted $\mathbb{P}_1 \dot{+} \mathbb{P}_2$, where $\mathbb{P}_1$ and $\mathbb{P}_2$ are components of an oriented path). For an oriented path $\mathbb{P}$ with a sequence of vertices $v_1, v_2, ..., v_k$, we refer to first vertex v_1 as $\alpha\mathbb{P}$ and the last vertex v_k as $\beta\mathbb{P}$.

Let $\mathbb{A} = (A, R)$ be a relational structure and let k be the arity of R. For each subset $L \subseteq [k]$ and element $l \in [k]$, the oriented path component $\mathbb{Q}_{L,l}$ was defined as a single edge if $l \in L$ and a zigzag if $l \in [k] \setminus L$. The oriented path $\mathbb{Q}_I$ was defined as

$$\mathbb{Q}_L = \bullet \longrightarrow \bullet \dot{+} \mathbb{Q}_{L,1} \dot{+} \mathbb{Q}_{L,2} \dot{+} ... \dot{+} \mathbb{Q}_{L,k} \dot{+} \bullet \longrightarrow \bullet$$

which has level $k+2$; one for each $\mathbb{Q}_{L,l}$ and two more to account for the single edges at each end. Having established these components, the digraph $\mathcal{D}(\mathbb{A})$ was defined.

Definition 4.2 ([8])**.** For every $\mathbf{e} = (a, \mathbf{r}) \in A \times R$, define the oriented path $\mathbb{P}_{\mathbf{e}} = \mathbb{Q}_{\{i: a = \mathbf{r}_i\}}$. The digraph $\mathcal{D}(\mathbb{A})$ is the relational structure $(A \cup R, A \times R)$ for which each edge relation $\mathbf{e} \in A \times R$ is replaced with the oriented path $\mathbb{P}_{\mathbf{e}}$ (with $a = \alpha\mathbb{P}_{\mathbf{e}}$ and $\mathbf{r} = \beta\mathbb{P}_{\mathbf{e}}$) and the vertex set $A \cup R$ does not include the $\bullet$ vertices used the make up each oriented path.

4.2 Bounded Width

For our particular CSP, and CSPs in general, we can use what are called *consistency checking* algorithms to identify and refute unsolvable instances without attempting to solve them using straightforward algorithms. Using the constraints already presented in the CSP and the variables in their scope, we can formulate "obvious" unary constraints (an assignment for a single variable) and use them to arrive at a contradiction. CSP subclasses whose instances with no solutions can all be refuted using consistency checking algorithms are said to have *bounded width.* More specific cases of bounded width can be classified based on the size of the constraints and number of variables used to create the new constraints.

Example 4.1. Consistency checking can be used to refute unsolvable cases of 2-Colourability. Suppose we are given a graph G which is a 5-cycle with vertices $v_1, v_2, ..., v_5$. The CSP instance can be expressed as $\mathcal{P} = (V, A, \mathcal{C})$ where: $V = \{v_1, v_2, ..., v_5\}$, $A = \{0, 1\}$, and $C_i = \{(v_i, v_{i+1}), \{(0, 1), (1, 0)\}\} \in \mathcal{C}$ for all $i \in [5]$ with addition modulo 5. By assigning a value to each vertex based on its colour, we are essentially presented with the following inequalities:

$$v_1 \neq v_2, \quad v_2 \neq v_3, \quad v_3 \neq v_4, \quad v_4 \neq v_5, \quad v_5 \neq v_1.$$

If take the first two inequalities $v_1 \neq v_2$ and $v_2 \neq v_3$, then we have $v_1 = v_3$ which introduces a new constraint $X = \{(v_1, v_3), \{(0, 0), (1, 1)\}\}$. If we take $v_1 = v_3$ and the third inequality $v_3 \neq v_4$, then we have $v_1 \neq v_4$ which introduces another new constraint $Y = \{(v_1, v_4), \{(0, 1), (1, 0)\}\}$. If we take $v_1 \neq v_4$ and the fourth inequality $v_4 \neq v_5$, then we have $v_1 = v_5$ which introduces another new constraint $Z = \{(v_1, v_5), \{(0, 0), (1, 1)\}\}$. Our new equality $v_1 = v_5$ contradicts the fifth inequality $v_5 \neq v_1$ and the constraint set $\mathcal{C} + \{X, Y, Z\}$ contains two contradicting constraints, C_5 and Z, which cannot both be satisfied. Hence, by using consistency checking, we are able to refute the CSP instance $\mathcal{P} = (V, A, \mathcal{C} + \{X, Y, Z\})$ and declare it unsolvable.

Every unsolvable instance of 2-Colourability can be refuted with a consistency checking algorithm like the one presented in Example 4.1 because graphs with odd cycles are not bipartite (see next paragraph). Hence, 2-COLOURABILITY has bounded width. More specifically, since the consistency checking algorithm creates 2-ary constrains using other constraints which, together, consist of at most 3 distinct variables, we say that 2-COLOURABILITY has $(2, 3)$-*width*. In order to provide a more mathematical definition of bounded width, we must first introduce the notion of minimality.

Definition 4.3 ([4]). A CSP instance $\mathcal{P} = (V, A, \mathcal{C})$ with binary constraints is $(2, 3)$-*minimal* if it contains a unique unary constraint C_x for all $x \in V$ and a unique binary constraint $C_{x,y}$ for all distinct $x, y \in V$ such that:

(1) For every pairwise distinct variables $x, y, z \in V$ and every binary relation $(a, b) \in C_{x,y}$, there exists a unary relation $(c) \in C_z$ such that $(a, c) \in C_{x,z}$ and $(b, c) \in C_{y,z}$ for all $a, b, c \in A$.

(2) For any pairwise distinct variables $x, y \in V$, we have $C_{x,y} = C_{y,x}^{-1}$ and the projection of $C_{x,y}$ onto the first and second coordinates are equal to C_x and C_y respectively.

To understand this definition, it is helpful to visualize a $(2, 3)$-minimal instance of a CSP as an "n-partite" graph. In this context, we will define a n-partite graph as a graph $G = (V, E)$ such that the vertex set $V(G)$ can be partitioned into disjoint sets $V_1, V_2, ..., V_n$ and edges can only exist between vertices in distinct sets. Our n-partite graph for a $(2, 3)$-minimal instance $\mathcal{P} = (V, A, \mathcal{C})$ will have $n = |V|$ partitions (one for each variable). In the partition corresponding to a variable $x \in V$, we will place a vertex for each relation $(a) \in C_x$. We will place an edge between vertices a and b if they are in separate partitions and $(a, b) \in C_{x,y}$. The first condition of Definition 4.3 guarantees that each edge in our graph belongs to a triangle. It is easy to see that a solution to our CSP instance corresponds to an n-clique in our n-partite graph for which each vertex is contained in a separate partition.

It is known that any binary CSP instance $\mathcal{P} = (V, A, \mathcal{C})$ can be converted to a $(2, 3)$-minimal instance in polynomial time as long as the unary constraint C_x is not empty for all $x \in V$ [**4**]; we will refer to the algorithm which performs this conversion as the $(2, 3)$-*minimality algorithm*. We are now ready to provide a proper definition of bounded width [**4**].

Definition 4.4 ([**4**])**.** A constraint language Γ is said to have $(2, 3)$-*width* if the $(2, 3)$-minimality algorithm refutes all unsolvable instances of CSP(Γ).

There are many other ways to define bounded width using equivalent characteristics. Like the tractability of a CSP, bounded width can be assessed using algebraic properties of the algebra of polymorphisms of the associated relational structure.

Definition 4.5. Let A be an algebra with universe A. An n-ary operation σ is called a *weak near-unanimity* (WNU) operation if for all $x, y \in A$,

(1) $\sigma(x, x, ..., x) = x$ (σ is *idempotent*), and

(2) $\sigma(y, x, x, ..., x) = \sigma(x, y, x, x, ...x) = \sigma(x, x, ..., x, y)$.

The presence of WNU operations in the algebra associated with a constraint language has a direct link to the solvability of a CSP. It was shown through a combination of results [**1**] that if the algebra associated with a constraint language Γ does not contain a WNU operation, then the subclass CSP(Γ) is **NP**-complete. The presence of WNU operations is also linked to whether a CSP has bounded width.

Theorem 4.2 ([**1**])**.** Let A_Γ be the algebra associated a constraint language Γ. The subclass CSP(Γ) has bounded width if and only if A_Γ contains two WNU operations with consecutive arities.

Example 4.2. Let $\mathbb{B} = (B, R)$ be a binary relational structure such that $B = \{0, 1\}$ and has the relation $0 \preceq 1$. The algebra of polymorphisms Pol($\mathbb{B}$) will contain (among others) the 2-ary operation $\vee$ ("or") such that $\vee(x, y) = \vee(y, x) = 1$ if and only if $x = 1$ or $y = 1$, and the 3-ary operation ω (majority) such that $\omega(x, y, z) = 1$ if and only if at least two of the numbers x, y, z are equal to 1 for all $x, y, z \in B$. We can see that $\vee$ is a WNU operation since:

(1) $\vee(0, 0) = 0$,
 $\vee(1, 1) = 1$, and

(2) $\vee(1, 0) = \vee(0, 1) = 1$.

Also, we can see that ω is a WNU operation since:

(1) $\omega(0, 0, 0) = 0$,
 $\omega(1, 1, 1) = 1$, and

(2) $\omega(1,0,0) = \omega(0,1,0) = \omega(0,0,1) = 0,$

$\omega(0,1,1) = \omega(1,0,1) = \omega(1,1,0) = 1.$

Since both $\vee$ and ω are WNU operations with consecutive arities, we can conclude that the subclass CSP($\mathbb{B}$) has bounded width.

4.3 Subalgebras of the Binary Constraint Language

The balanced digraph $\mathbb{A}$ created as a result of reducing a relational structure whose algebra of polymorphisms contains few subpowers has binary edge relations. As a result, the subclass CSP($\mathbb{A}$) will have binary constraint relations. In this section, we will study the subalgebras of the algebra of polymorphisms $\mathsf{A}_{\mathbb{A}}$. The subalgebras of $\mathsf{A}_{\mathbb{A}}$ can be interpreted as compatible unary relations thus making them definable primitive positive formulas over basic relations and constants. In fact, the primitive positive formulas can be defined using relations only because we can omit the constants since every constant corresponds to basic unary relations. Since $\mathsf{A}_{\mathbb{A}}$ contains a k-edge operation, its minimal subalgebras can only be of Type 2, 3, or 4 in the sense of Tame Congruence Theory [**16**] (not to be confused for similarity type for relational structures). Of these algebra types, only Type 3 and Type 4 have what we call an absorbing element for each binary operation.

Definition 4.6. Let $\mathsf{A} = (A, \mathcal{F})$ be an algebra and let $* \in \mathcal{F}$ be a binary operation. An element $x \in A$ is called an *absorbing element* if for all $a \in A$, we have $x * a = a * x = x$.

Example 4.3. Let $\mathsf{A} = (\mathbb{Z}, \{\text{gcd}, \times\})$ be an algebra whose universe is the set of integers and whose binary operations include the Greatest Common Divisor and product operations. We can see that the element $1 \in \mathbb{Z}$ is an absorbing element for gcd since $\text{gcd}(1, x) = \text{gcd}(x, 1) = 1$ for all $x \in \mathbb{Z}$. Similarly, we can see that the element $0 \in \mathbb{Z}$ is an absorbing element for $\times$ since $0 \times x = x \times 0 = 0$ for all $x \in \mathbb{Z}$.

Type 2 (affine type): Suppose M is a strictly simple subalgebra of $\mathsf{A}_{\mathbb{A}}$ of Type 2; it is a strictly simple affine module [**18**]. An algebraic structure is said to be *strictly* simple if it is simple, its domain has at least 2 elements, and it has no proper, non-trivial subalgebra. An algebra M is said to be affine with respect to an Abelian group $\widehat{\mathsf{M}} = (M, \{+, -, 0\})$ if M and $\widehat{\mathsf{M}}$ have the same universe, $m(x, y, z) = x - y + z$ is a term operation of M, and the 4-ary relation $Q_{\widehat{\mathsf{M}}} = \{(a, b, c, d) \in M^4 \mid a - b + c = d\}$ is a subuniverse of M^4. For a vector space $\mathsf{K}_{\widehat{\mathsf{M}}}$, let $T(\mathsf{K}_{\widehat{\mathsf{M}}})$ denote the group, $\{x + m \mid m \in M\}$, of translations of $\mathsf{K}_{\widehat{\mathsf{M}}}$, and let $C(\mathsf{K}_{\widehat{\mathsf{M}}})$ denote the set of all binary operations $rx + (1 - r)y$, where $r \in \text{End}(\mathsf{K}_{\widehat{\mathsf{M}}})$, the set of all endomorphisms of $\widehat{\mathsf{M}}$ (also known as the endomorphism ring of $\widehat{\mathsf{M}}$). A finite algebra, M with universe M is said to be *affine* if and only if there exists a finite field K, a vector space $\mathsf{K}_{\widehat{\mathsf{M}}} = (M, +, \mathsf{K})$, and an endomorphism e of $\mathsf{K}_{\widehat{\mathsf{M}}}$ with $e^2 = e$ such that M is term-equivalent to one of the algebras

$$(M, \{x - y + z, C(\mathsf{K}_{\widehat{\mathsf{M}}}), e\}) \quad \text{or} \quad (M, \{x - y + z, C(\mathsf{K}_M), e, T(\mathsf{K}_{\widehat{\mathsf{M}}})\})[\mathbf{18}].$$

Every finite simple affine algebra has, as it underlying universe, a finite-dimensional vector space over a finite field F with universe F, where $|F| = p^k$ for some prime number p and positive integer k.

Definition 4.7. A *cyclic group* $\mathsf{C} = (C, *)$ is an Abelian group with an invertible associative operation $*$ whose universe contains an element which can generate the entire set.

Example 4.4. The group $\mathbb{Z}_n = (\{0, 2, ..., n-1\}, +)$ under addition modulo n is a cyclic group. The universe of $\mathbb{Z}_n$ can be generated by the element 1 since for all $k \in \{0, 2, ..., n-1\}$, we have $k = 1^k$, where the notation 1^k represents the addition of 1 to itself k times. Here, we have $0 = 1^n$ since $0 = n \pmod{n}$.

We consider the following theorem known as the Fundamental Theorem of Finite Abelian Groups,

Theorem 4.3. Suppose G is an Abelian group. Then G can be expressed uniquely as a direct product of finite cyclic groups of order p^k for some prime number p and some positive integer k.

If we fix an element $0 \in M$ in the universe of M such that $x + y = x - 0 + y$, our theorem implies that under the binary operation $+$, the subuniverse M is a direct sum of a finite number of cyclic groups $\mathbb{Z}_{p^r}$ for some $r \in [k]$.

Type 3 (Boolean Type): Suppose M is a strictly simple subalgebra of $\mathsf{A}_{\mathbb{A}}$ of Type 3. We consider three kinds: quasiprimal, Boolean, and "cross-type" [**18**]. The quasiprimal algebras are algebras whose operations are all induced by a polymorphism on some relation on the elements of its universe. Hence, every element of a quasiprimal universe is an absorbing element [**18**]. A Boolean algebra $\mathsf{M} = (M, \mathcal{F})$ is one for which the universe $M = \{\text{TRUE}, \text{FALSE}\}$ contains the Boolean truth values and the set $\mathcal{F} = \{\wedge, \vee, \neg\}$ contains the binary $\wedge$ ("and") and $\vee$ ("or") operations and the unary $\neg$ ("not") operation. It is also true that every element in a Boolean algebra is an absorbing element since TRUE is the absorbing element for $\vee$, and FALSE is the absorbing element for $\wedge$.

To define a "cross-type" subalgebra M of $\mathsf{A}_{\mathbb{A}}$, we must define a new type of group.

Definition 4.8. A *permutation group* is a group $\mathbb{G} = (G, \circ)$ such that the elements of G are permutations $p : P \longrightarrow P$ of some finite set P and the operation $\circ$ is the composition of permutations such that for $f(P), g(P) \in G$, we have $f \circ g = f(g(P))$.

We introduce the following notations regarding $\mathsf{M} = (M, \mathcal{F})$. For $m \in M$ and integer $k \geq 2$, we define the following k-ary relation [**18**]

$$X_k^m = \{(m_1, m_2, ..., m_k) \in M^k : m_i = m \text{ for at least one } i \in [k]\}.$$

Also, we define $\mathcal{F}_k^m$ to be the set of operations f on universe M for which X_k^m is a subuniverse of the algebra $(M, f)^k$. Let G be a permutation group acting on M. We define $\mathcal{R}_M(\mathsf{G})$ to be the set of operations f on M which are idempotent and admit each member of G as an automorphism. A "cross-type" subuniverse is a strictly simple algebra $\mathsf{M} = (M; \mathcal{F}_k^m \cap \mathcal{R}_M(\mathsf{G}))$ [**18**].

Theorem 4.4. If M is a "cross-type" algebra, then it cannot contain a Mal'tsev operation.

Proof: We argue by contradiction. Suppose $\mathsf{M} = (M, \mathcal{F})$ is a "cross-type" algebra and $m(x, y, z) \in \mathcal{F}$ is a Mal'tsev operation on M. Let 0 be a distinct element used to define the "cross" and let $x \neq 0$. If the cross for M is k-dimensional with $k \geq 3$, we consider the following k-tuples,

$$\mathbf{a} = (0, x, x, x, ..., x), \;\; \mathbf{b} = (0, 0, x, x, ..., x), \;\; \mathbf{c} = (x, 0, x, x, ..., x).$$

Clearly, $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are in the k-cross and $m(\mathbf{a}, \mathbf{b}, \mathbf{c}) = (x, x, x, ..., x)$ since $m(0, 0, x) = m(x, 0, 0) = m(x, x, x) = x$. This k-tuple is also in the k-cross which implies that $x = 0$, a contradiction. Hence, a "cross-type" algebra cannot contain a Mal'tsev operation. $\square$

Before we continue, we must expand of the algebraic concepts discussed so far. Let $\mathsf{A} = (A, \mathcal{F})$ and $\mathsf{B} = (B, \mathcal{F})$ be algebras of the same type. We call B an *absorbing subalgebra* of A if there exists an idempotent term operation t in A such that $t(B, B, ..., B, A, B, B, ..., B) \subseteq B$. To be clear, if all but one of the arguments in t are elements in B, then t outputs an element in B. An algebra A is called *hereditarily absorption-free* (HAF) if each subalgebra $\mathsf{B} \leq \mathsf{A}$ has no absorbing subalgebra. Before stating our next theorem, we must define a new type of operation.

Definition 4.9. Let t be an n-ary operation. We call t a *Taylor term* if it is idempotent and, for each coordinate $i \leq n$, satisfies

$$t(x_1, x_2, ..., x_n) = t(y_1, y_2, ..., y_n),$$

where $x_1, x_2, ..., x_n, y_1, y_2, ..., y_n \in \{x, y\}$ and $x_i = x, y_i = y$.

Theorem 4.5 ([**3**])**.** Let A be a finite idempotent algebra with a Taylor term. If A is HAF, then it has a Mal'tsev term.

We will use the contrapositive statement to Theorem 4.5 to show that a "cross-type" algebra contains an absorbing element.

Theorem 4.6. Let M be a Type 3 algebra with universe M. If M is a "cross-type" algebra, then it must contain an absorbing element.

Proof: We argue by contradiction. Suppose M is a strictly simple "cross-type" algebra with no absorbing element. Since M cannot have a Mal'tsev operation, it is not HAF. So there exists some subalgebra of M which has an absorbing subalgebra. But, since M is an algebra on a binary set, the only proper subuniverse of M is single element. An algebra with a single element in its universe cannot have an absorbing subalgebra. Therefore, M must be HAF and, as a result, must have a Mal'tsev term; a contradiction. Hence, M must have an absorbing element. $\square$

Type 4 (lattice type): The only strictly simple subalgebra of $\mathsf{A}_{\mathbb{A}}$ of Type 4 is the 2-element lattice,

$$\mathsf{M} = (\{0, 1\}, \{\wedge, \vee\}).$$

Assuming that the elements are ordered $0 \preceq 1$, the element 0 is the absorbing element of the $\wedge$ ("meet") operation and the element 1 is the absorbing element of the $\vee$ ("join") operation.

Chapter 5

A BOOLEAN CIRCUIT ALGORITHM FOR CSPS WITH FEW SUBPOWERS

Having reduced our constraint template to one which is binary, the subalgebras of the algebra of polymorphisms of our relational structure allow us to define a solution set to a CSP with few subpowers as an algebra itself using the context of these subalgebras. We are then able to define the CSP as a logical formula which we can interpret as a Boolean tree. Using our Boolean tree interpretation, we are able combine pairs of constraints and eliminate inconsistent solutions until we reach a solution set which satisfies all of the constraints. In order to interpret our CSP solution as an algebra, we must introduce an algebraic structure similar to that of a subpower.

5.1 Subdirect Products of two Strictly Simple Subalgebras

In this section, we will construct and analyze an algebraic structure made up of the solution set of a CSP. Let $\mathcal{P} = (V, A, \mathcal{C})$ be a CSP instance and let $\mathcal{F}$ be the set of functions $f : V \longrightarrow A$ which make up the solution set of $\mathcal{P}$ (in the form of functions which map V to A). We can define a "solution algebra" $\mathsf{S} = (A, \mathcal{F})$ to be the algebra associated with solution set of $\mathcal{P}$. Before we continue, we must define a new type of subalgebra which is similar to a subpower.

Definition 5.1 ([**9**]). Let $\mathsf{A}_1, \mathsf{A}_2, ..., \mathsf{A}_n$ be a set of finite algebras with universes $A_1, A_2, ..., A_n$ respectively. An algebra B with universe B is a *subdirect product* of $\mathsf{A}_1, \mathsf{A}_2, ..., \mathsf{A}_n$ if it is a subalgebra of $\mathsf{A}_1 \times \mathsf{A}_2 \times ... \times \mathsf{A}_n$ and $\text{proj}_i B = A_i$ for all $i \in [n]$. We denote this as $\mathsf{B} \leq_{sp} \mathsf{A}_1 \times \mathsf{A}_2 \times ... \times \mathsf{A}_n$.

As per its definition, a subdirect product is a generalization of a subpower. Whereas a subpower is a subalgebra of the product of n copies of an algebra A, a subdirect product is a subalgebra of the product of n different algebras A_i. We will now prove a well-known property of subdirect products.

Theorem 5.1. Suppose A_1 and A_2 are subalgebras of $\mathsf{A} = (A, \mathcal{F})$. If $x \in \mathsf{A}$, then the set $[x]^+ = \{y \in A_2 \mid (x, y) \in A_1 \times A_2\}$ is a subuniverse of A_1.

Proof: Let $y_1, y_2, ..., y_k \in [x]^+$. Suppose we take some k-ary operation $f \in \mathcal{F}$. Since f is idempotent, we have $x = f(x, x, ..., x) \in A_1$, and since A_2 is a subalgebra of A, we have $y = f(y_1, y_2, ..., y_k) \in A_2$. Therefore, we have $(f(x, x, ..., x), f(y_1, y_2, ..., y_k)) \in A_1 \times A_2$ and $f(y_1, y_2, ..., y_k) \in [x]^+$. $\square$

The previous theorem leads to the following corollary presented by Burris and Sankappanavar.

Corollary 5.1 ([**9**]). Let A_1 and A_2 be strictly simple subalgebras of A with universes A_1 and A_2 respectively. A subdirect product of A_1 and A_2 must be either,

(1) be isomorphic to $\mathsf{A}_1 \times \mathsf{A}_2$,

(2) be equivalent to $\mathsf{A}_1 \times \mathsf{A}_2$ (a direct product), or

(3) have the following property: for every $x \in A_1$, either the set $[x]^+$ is a trivial subuniverse of A_2 or $[x]^+ = A_2$.

We are almost ready to present the main result of this paper. Before we design an algorithm which solves a CSP with few subpowers using a Boolean circuit, we must prove our theorem which establishes algebraic properties of the solution algebra crucial to the formulation of the Boolean circuit.

Theorem 5.2. Let $\mathcal{P} = (V, A, \mathcal{C})$ be an instance of CSP($\mathbb{A}$), where $\mathbb{A}$ is relational structure reduced to one which is binary as in Chapter 4.1 and altered using consistency checking as in Chapter 4.2 whose algebra of polymorphisms A has few subpowers. Let $\mathsf{S} = (A, \mathcal{F})$ be the solution algebra associated with the set of functions $\mathcal{F}$ which make up the solution set of $\mathcal{P}$. The following statements hold true:

(1) S is a subalgebra of A^n, where $n = |V|$ (S is a subpower of A).

(2) There exists a subalgebra B of S such that

$$\mathsf{B} \leq_{sp} \mathsf{B}_1 \times \mathsf{B}_2 \times ... \times \mathsf{B}_n,$$

where $\mathsf{B}_i \leq \mathsf{A}$ for all $i \in [n]$ are either trivial or strictly simple affine modules.

Proof: (1) follows directly from the definition of a solution set for a CSP and the definition of a polymorphism from Chapter 2; we will prove (2). It is clear from (1) that the solution algebra S can be written as a subdirect product

$$\mathsf{S} \leq_{sp} \mathsf{S}_1 \times \mathsf{S}_2 \times ... \times \mathsf{S}_n \leq \mathsf{A}_{x_1} \times \mathsf{A}_{x_2} \times ... \times \mathsf{A}_{x_n} \leq \mathsf{A}^n,$$

such that every S_i is a subalgebra of A_{x_i}, which we will define as the subalgebra of A whose universe consists solely of the element x_i for all $i \in [n]$. Suppose, without a loss of generality, that S_1 is neither trivial nor a strictly simple affine module. Then S_1 must itself contain a strictly simple subalgebra M_1. If S_1 is in fact trivial or a strictly simple affine module, then set $\mathsf{S}_1 = \mathsf{M}_1$. Let the algebra $\widehat{\mathsf{S}}_1 = \text{proj}_1\mathsf{S}$ be the restriction of S to those solution n-tuples whose first projection coordinate lies in the universe of M_1. Proceeding iteratively on all S_i up to $i = n$, we have

$$\text{proj}_n\mathsf{S} = \widehat{\mathsf{S}}_n \leq_{sp} \mathsf{M}_1 \times \mathsf{M}_2 \times ... \times \mathsf{M}_n,$$

where M_i are all trivial or strictly simple and $\widehat{\mathsf{S}}_n$ is a subalgebra of S. However, strictly simple subalgebras of Type 3 and Type 4 have absorbing elements as explained in Chapter 4. Applying absorption in each coordinate i such that M_i is neither trivial nor a strictly simple affine module gives us the desired result. Hence, by setting $\widehat{\mathsf{S}}_n = \mathsf{B}$ and $\mathsf{M}_i = \mathsf{B}_i$, we have proven our claim. $\square$

5.2 Constructing a Logical Formula Capturing the CSP

Let Γ be a constraint language with few subpowers and let A_Γ be the associated algebra. Suppose $\mathcal{P} = (V, A, \mathcal{C})$ is an instance of CSP(Γ) with binary constraints as defined in Theorem 5.2. We will construct a logical formula which can solve our CSP. We may also assume that $\mathcal{P}$ is a simplified CSP such that every pair of variables in V appears as the scope of, at most, one constraint. If we set $n = |V|$, then there are n^2 possible constraints in $\mathcal{C}$. For some integer l, where $1 \leq l \leq n^2$, the constraints of $\mathcal{P}$ can be written as $C_i(x_{i_1}, x_{i_2})$ for all $1 \leq i \leq l$ and distinct $x_{i_1}, x_{i_2} \in V$. We can now interpret the solvability of $\mathcal{P}$ as being equivalent to the satisfiability of the primitive positive formula

$$\exists x_1, x_2, ..., x_n \bigwedge_{i=1}^{l} C_i(x_{i_1}, x_{i_2}).$$

This formula can be arranged as a Boolean tree $\mathcal{B}$ with leaves labeled "C_i" and internal vertices labeled "$\wedge$". It is clear that the total number of leaves (input gates) in $\mathcal{B}$ is of order $\mathcal{O}(n^2)$ and the depth of $\mathcal{B}$ is of order $\mathcal{O}(\log n)$ since $\log l \leq \log(n^2) = 2\log n$.

Next, we will analyze the complexity of the Boolean formulas corresponding to the leaves in $\mathcal{B}$. Every constraint $C_i(x_{i_1}, x_{i_2})$ can be written as the disjunction

$$\bigvee_{\mathsf{M}_1, \mathsf{M}_2 \leq \mathsf{A}_\Gamma} \big(\phi_{M_1}(x_{i_1}) \wedge \phi_{M_2}(x_{i_2}) \wedge \psi_{M_1,M_2}(x_{i_1}, x_{i_2})\big),$$

where the disjunction ranges over all pairs, M_1 and M_2, of trivial and strictly simple subalgebras of A. The formulas ϕ_{M_1} and ϕ_{M_2} are primitive positive formulas which define the subuniverses M_1 and M_2 of M_1 and M_2 respectively while the formula ψ_{M_1,M_2} states that there exists a subdirect product of $\mathsf{M}_1 \times \mathsf{M}_2$ which is contained in the solution set of $\mathcal{P}$. The number of pairs, M_1 and M_2, of trivial strictly simple subalgebras of A_Γ is a constant dependent only on its universe and operations and not on the number of variables $n = |V|$; it will not affect the complexity of solving $\mathcal{P}$.

We will closely study the formula ψ_{M_1,M_2}. It will vary based on whether at least one of the subalgebras M_1 or M_2 does not has a Mal'tsev operation. If both M_1 and M_2 have Mal'tsev operations, their subdirect product must either be an isomorphism of $\mathsf{M}_1 \times \mathsf{M}_2$ or a direct product according to Corollary 5.1. In the case of an isomorphism, ψ_{M_1,M_2} may be written as $x_{i_2} = x_{i_1} + a$, where a is some constant in the Abelian group which is the isomorphism type of both M_1 and M_2. In fact, as discussed earlier in this chapter, this equation for ψ_{M_1,M_2} is equivalent to the conjunction of cyclic equations over $\mathbb{Z}_{p^r}$ for some prime number p and a suitable positive integer r. In the case of a direct product, the formula for ψ_{M_1,M_2} can be omitted from the constraint. If at least one of M_1 and M_2 has no Mal'tsev operation, then the subalgebra must contain an absorbing element as explained in Chapter 4.3. We have a direct product of two subalgebras where at least one is trivial. Like in the previous case, the formula for ψ_{M_1,M_2} can be omitted from the constraint while the formulas for ϕ_{M_1} and ϕ_{M_2} can be changed to one for a trivial subalgebra corresponding to the absorbing element.

We are ready to state and prove our main result and show that the subclass of CSPs with few subpowers belongs to the class $\mathbf{NC}^2$. As mentioned earlier, the constraints of $\mathcal{P}$ can be arranged as a Boolean tree with each leaf being labeled as a constraint and each internal vertex being labeled "$\wedge$". We will show that the formulas corresponding to each $\wedge$ vertex can be computed in logspace.

Theorem 5.3. Let $\mathcal{P} = (V, A, \mathcal{C})$ be an instance of CSP($\mathbb{A}$), where $\mathbb{A}$ is relational structure reduced to one which is binary as in Chapter 4.1 and altered using consistency checking as in Chapter 4.2 whose algebra of polymorphisms A has few subpowers, and let $V = \{x_1, x_2, ..., x_n\}$. Suppose $\mathcal{P}$ is formulated as a Boolean tree $\mathcal{B}$ as described so far in this section. Given the formulas corresponding to its children vertices, the formula corresponding to a $\wedge$ vertex can be computed in logspace.

Proof: Inductively, we may assume that the formulas for the children vertices of each $\wedge$ vertex can be expressed as

$$\bigvee_{C_i \in \mathcal{C}}^{n} \big(\theta_1(x_1) \wedge \theta_2(x_2) \wedge ... \wedge \theta_n(x_n) \wedge \mathrm{sys}(x_1, x_2, ..., x_n)\big),$$

where each $\theta_i(x_i)$ is a primitive positive formula defining a trivial or strictly simple subalgebra of A whose universe consists solely of the element x_i for all $i \in [n]$, and $\mathrm{sys}(x_1, x_2, ..., x_n)$ is the formula which describes a conjunction of linear systems of cyclic equations. The conjunction of two such disjunction terms which appear in the children vertices of the $\wedge$ vertex can be computed in logspace. If the vertices are inconsistent, the conjunction of the two terms will be deleted from the disjunction. Therefore, the formula corresponding to the root vertex of the binary tree can be computed in logspace. It is known that any system of linear equations over a finite field can be computed in $\mathbf{NC}^2$ [**20**]. If we add the depth of the tree to our overall algorithm, it will be of order $\mathcal{O}(\log^2 n + 2\log n) = \mathcal{O}(\log^2 n)$. $\square$

Having shown that the formulas in the Boolean circuit $\mathcal{B}$ can be solved in logspace, and that the circuit itself has logarithmic depth and polynomially many input gates, we can conclude that the subclass of CSPs with few subpowers belongs to the class $\mathbf{NC}^2$.

Chapter 6

CONCLUDING REMARKS AND OPEN PROBLEMS

We conclude by providing a brief overview of the material covered in this paper. We began by summarizing the full scope of the constraint languages whose CSP subclasses can be solved using the "robust Gaussian-like" algorithm described by Idziak et al [**15**]. The algorithm they described represents a full generalization of the algorithms presented by Bulatov and Dalmau in [**7**] and [**10**] among other contributions. Although we eventually go on to explore alternative methods to solve CSPs with few subpowers, we still consider [**15**] to be the primary source on such CSPs. Along with the conjectures we will present, there are many more open problems pertaining to the study of CSPs. Our result in this paper does not solve any of them, but it does tighten the result presented in [**15**] by placing the subclass of CSP with few subpowers in the class $\mathbf{NC}^2$, a subset of $\mathbf{P}$.

6.1 A Brief Summary

There exists an alternative algorithm more efficient than the one presented in [**15**] which solves CSPs with few subpowers. The algorithm we describe solves these CSPs using a Boolean circuit placing the subclass of CSPs with few subpowers in the class $\mathbf{NC}^2$. First, our method requires that the CSP be reduced to one which has only binary constraints so that the algebra of polymorphisms of the associated binary relational structure yields favorable subalgebras which will be used to define the CSP as a logical formula. Afterwards, we must run a consistency checking algorithm to use the constraints already provided to create new ones to narrow down our set of possible solutions.

Once our CSP has been properly reduced and modified, we set the stage for our main result by presenting and proving Theorem 5.2, which interprets our solution set as an algebra itself. Using this interpretation, we are able to define our CSP as a primitive positive formula which can be arranged as a Boolean tree where each leaf is a constraint. The process of solving the CSP is akin to traveling up our binary tree by combining pairs of constraints and eliminating inconsistent functions from the solution set until we have reached the root of the tree. The remaining functions in the solution set will be those which satisfy all of the constraints. Since there are polynomially many constraints and the depth of the tree is logarithmic, we can place the subclass of CSPs with few subpowers in the class $\mathbf{NC}^2$.

6.2 Open Problems

We conclude our exposition on the subject of tractable CSP subclasses by examining some of the open problems related to the subject of this paper. Like the Dichotomy Conjecture, which remains open decades after its conception, the following conjectures remain open as of this writing.

Conjecture 6.1. Every CSP over a finite template with few subpowers can be reduced in polynomial time (or logspace) to one which has bounded width. Furthermore, the reduced CSP has only binary constraints.

In [**1**] a polynomial time reduction to a so-called *syntactically simple binary instance* is presented; the reduction can be carried out in such a way that the obtained binary instance is (2,3)-minimal. This reduction can be viewed as a counterpart to our reduction of a CSP to a digraph since our reduction is to instances of affine domains which lack the properties associated with domains of bounded width. For a brief introduction to Polynomial Interpretation Logic and its role in the problem of solving CSPs based on finite relational structures in polynomial time, we refer the reader to [**12**].

Conjecture 6.2. Every CSP over a finite template with few subpowers is definable in Polynomial Interpretation Logic with Cardinality Quantifiers (or, equivalently, in the Choiceless Polynomial Time with Counting).

If this conjecture were to be confirmed, it would provide evidence that the two logics, which are equal in their expressive power, can fully encapsulate the subclass of tractable CSPs based on finite relational structures. This would also provide evidence that the Dichotomy for Finite Template CSPs can be logically formulated. The problem of expressibility of finite template CSPs in

Choiceless Polynomial Time with Counting is also closely related to expressibility of certain known tractable cases of the graph isomorphism problem in the same logic.

Conjecture 6.3. (**High Parallelizability Dichotomy**) A CSP subclass is in the complexity class **NC** if and only if the variety generated by its algebra of polymorphisms omits Type 5 (semi-lattice type) algebras in the sense of Tame Congruence Theory [**16**].

BIBLIOGRAPHY

[1] Barto, L. and Kozik, M. *Constraint Satisfaction Problems of Bounded Width*. Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS'09, 2009, 595-603.

[2] Barto, L.; Kozik, M.; and Niven, T. *The CSP Dichotomy Holds for Digraphs with no Source and no Sinks (A Positive Answer to a Conjecture of Bang-Jensen and Hell)*. SIAM Journal on Computing, 2009, 38, 1782-1802.

[3] Barto, L.; Kozik, M.; and Stanovský, D. *Mal'tsev Conditions, Lack of Absorption, and Solvability*. Algebra Universalis, 2015, 74, 185-206.

[4] Barto, L.; Krokhin, A.; and Willard, R. *Polymorphisms, and How to Use Them The Constraint Satisfaction Problem: Complexity and Approximability*. [Result of a Dagstuhl Seminar], 2017, 1-44.

[5] Berman, J.; Idziak, P.; Marković, P.; McKenzie, R.; Valeriote, M.; and Willard, R. *Varieties with Few Subalgebras of Powers*, Transactions of the American Mathematical Society, 2010, 362, 1445-1473

[6] Bulatov, A. *A Dichotomy Theorem for Constraint Satisfaction Problems on a 3-Element Set*. Journal of the ACM, 2006, 53, 66-120.

[7] Bulatov, A. and Dalmau, V. *A Simple Algorithm for Mal'tsev Constraints*. SIAM Journal on Computing, 2006, 36, 16-27.

[8] Bulín, J.; Delić;, D.; Jackson, M.; and Niven, T. *A Finer Reduction of Constraint Problems to Digraphs.* Logical Methods in Computer Science, 2015, 11, 1-33.

[9] Burris, S. and Sankappanavar, H. *A Course in Universal Algebra.* Springer-Verlag, 1981.

[10] Dalmau, V. *Generalized Majority-Minority Operations are Tractable.* Logical Methods in Computer Science, 2006, 2, 1-15.

[11] Feder, T. and Vardi, M. *The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory.* SIAM Journal on Computing, 1999, 28, 57-104.

[12] Grädel, E.; Kaiser, ; Pakusa, W.; and Schalthöfer, S. *Characterising Choiceless Polynomial Time with First-Order Interpretations.* Proceedings of 30th Annual ACM/IEEE Symposium on Logic in Computer Science, 2015.

[13] Habte, A. *Constraint Satisfaction Problems in the Logic LFP+Rank.* Ryerson University, 2015.

[14] Hell, P. and Něsetřil, J. *On the Complexity of H-Coloring.* Journal of Combinatorial Theory, 1990, Ser. B 48, 92-110.

[15] Idziak, P.; Marković, P.; Mckenzie, R.; Valeriote, M.; and Willard, R. *Tractability and Learnability Arasing from Algebras with Few Subpowers.* SIAM Journal on Computing, 2010, 39, 3023-3037.

[16] Larose, B. and Tesson, P. *Universal Algebra and Hardness Results for Constraint Satisfaction Problems.* Theoretical Computer Science, 2009, 410, 1629-1647.

[17] Schaefer, T. *The Complexity of Satisfiability Problems.* Conference Record of the Tenth Annual ACM Symposium on Theory of Computing, 1978, 216-226.

[18] Szendrei, Á. *A Survey of Strictly Simple Algebras and Minimal Varieties.* Universal Algebra and Quasigroup Theory, 1992, 19, 209-239.

[19] Valeriote, M. *A Subalgebra Intersection Property for Congruence Distributive Varieties.* Canadian Journal of Mathematics, 2009, 61, 451-464.

[20] Vollmer, H. *Introduction to Circuit Complexity*, Springer-Verlag, 1999.